Parallel and Distributed Tools for Evolutionary Computations

by ***Marc Parizeau***, professor
Dep. of Electrical and Computer Engineering,
Computer Vision and Systems Laboratory,
Université Laval
and
Deputy Director of CLUMEQ

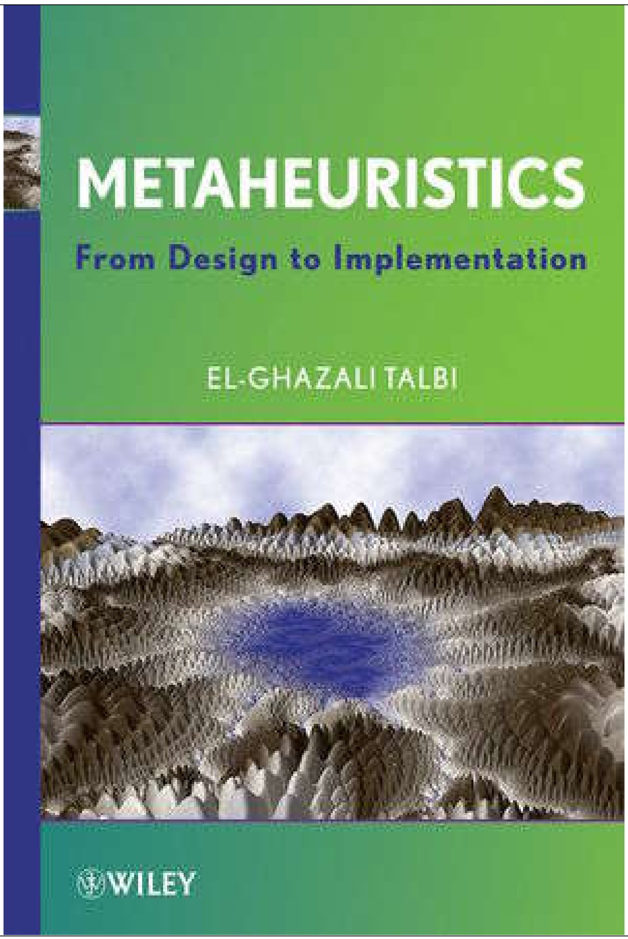
Outline

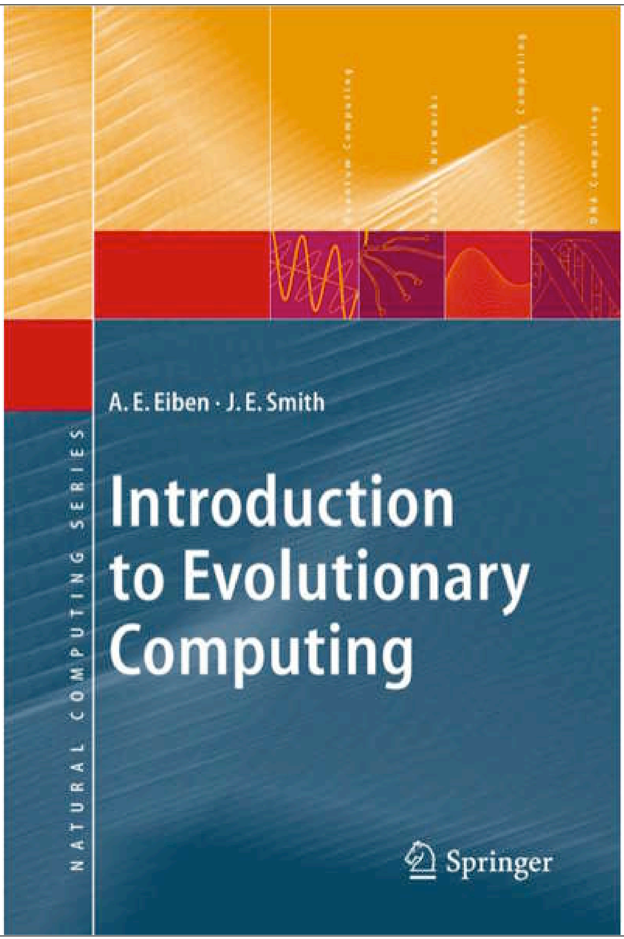
- Part I: ***fundamentals***
- Part II: ***tools***
  - ✓ hardware: *Colossus*
  - ✓ software
  - ✓ Open BEAGLE
- Part III: ***architecture***
  - ✓ *Distributed Task Manager* (DTM)
  - ✓ *Evolutionary Algorithms in Python* (EAP)
  - ✓ DTM+EAP = DEAP computing!

Part I: ***fundamentals***

- Evolutionary computations for artificial intelligence?
- Flavours of evolutionary Algorithms
- Multiobjective optimization
- Parallelism

An excellent book that covers metaheuristics in general, including evolutionary algorithms...



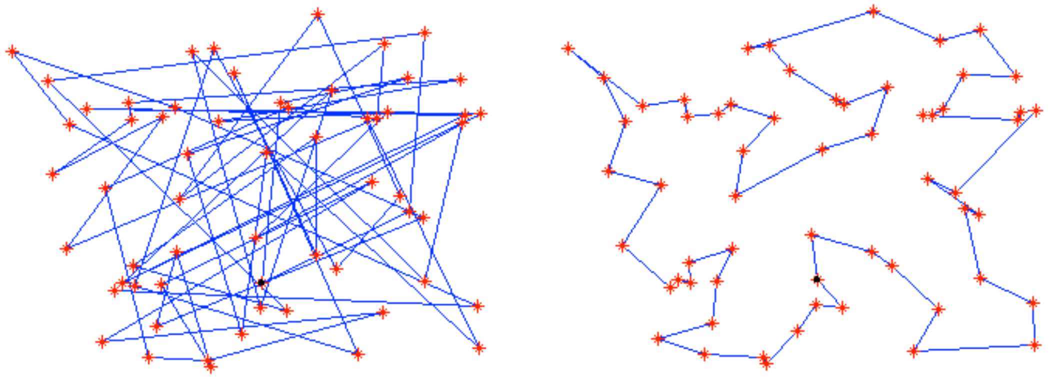

A. E. Eiben · J. E. Smith
Introduction to Evolutionary Computing
NATURAL COMPUTING SERIES
Springer

Another good book that covers everything that you want to know about evolutionary algorithms...

Why should you care?

- Optimization problems are everywhere
- Computing optimal solutions is often intractable
  - ✓ thus the need for approximate optimization methods that generate "*acceptable*" solutions in a "*reasonable*" amount of time
- Evolutionary Algorithms (EA) are good approximate problem solving methods
  - ✓ generic in nature
  - ✓ efficient for hard problems

Example 1
Traveling salesman

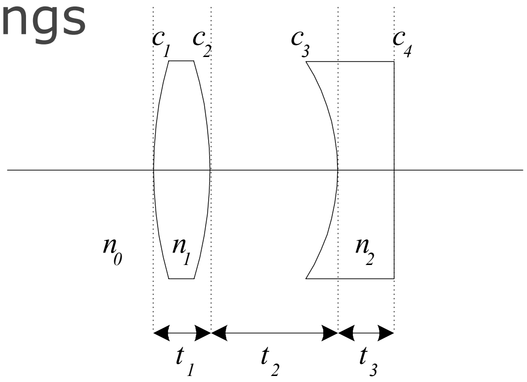


problem: finding the shortest «*hamiltonian cycle*» ?

> 10^{81} possibilities (for 60 cities)

Example 2
Lens system design

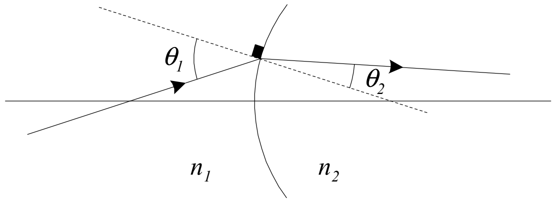
- Lens systems are very much non-linear
- Design parameters include number of lenses, curvature, refractive indices, and spacings



c: curvature
n: refractive index
t: spacing

- Modelling should be based on the Snell-Descartes formula:

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$



- but, instead, uses the first order paraxial approximation that assumes light rays close to the optical axes:

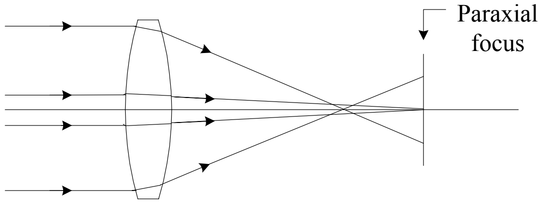
$$\sin\phi = \phi - \frac{\phi^3}{3!} + \frac{\phi^5}{5!} - \cdots$$

$$\text{let } \phi \approx 0 \Longrightarrow \sin\phi \approx \phi.$$

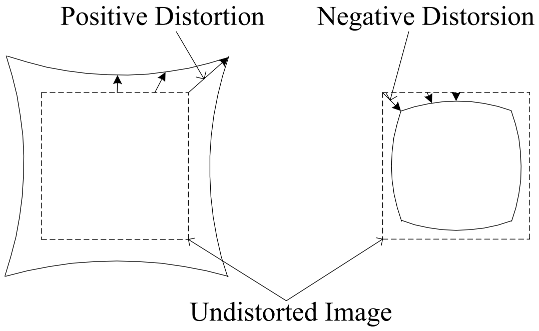
$$\boxed{n_1\theta_1 \approx n_2\theta_2}$$

- The five Seidel aberrations results from the difference between third and first order optics: *spherical*, *coma*, *astigmatism*, *field curvature*, and *distortion*.

$$\sin\phi = \phi - \frac{\phi^3}{3!} + \frac{\phi^5}{5!} - \cdots$$



spherical aberration



Christian Gagné, Julie Beaulieu, Marc Parizeau and Simon Thibault, "**Human-Competitive Lens System Design with Evolution Strategies**", Applied Soft Computing, September 2008.

Example 3
Surveillance and protection

- For sensor networks
- Optimizing sensor placement to:
  - ✓ maximize coverage
  - ✓ minimize cost
  - ✓ minimize intervention time
- Integrate with:
  - ✓ sensor models
  - ✓ geographical information systems

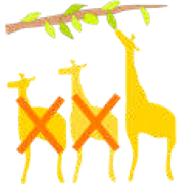
Part I: *fundamentals*

- Evolutionary computations for artificial intelligence?
- Flavours of evolutionary Algorithms
- Multiobjective optimization
- Parallelism

Evolutionary algorithms

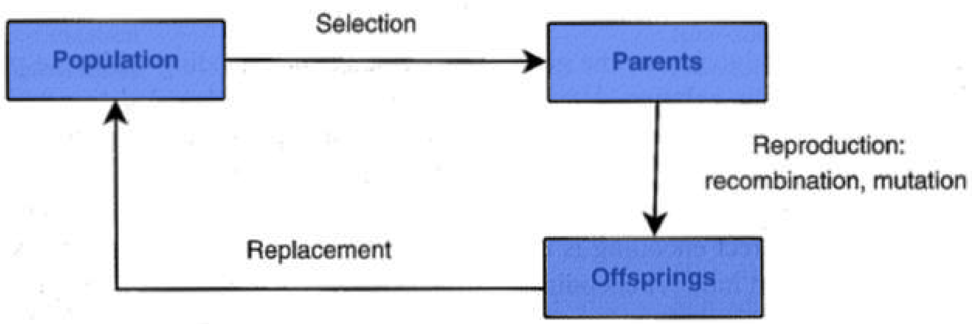
- EAs are *population* based *metaheuristics* that can solve most any optimization problem
- They come in many flavours, including the following:
  - ✓ Genetic Algorithms (GA)
  - ✓ Evolutionary Strategies (ES)
  - ✓ Evolutionary Programming (EP)
  - ✓ Genetic Programming (GP)

Darwin theory



- **Natural selection** is the process by which *heritable traits* that make it more likely for an organism to *survive* and successfully *reproduce* become more common in a population over *successive generations*. It is a key mechanism of evolution.

High-level template



generational evolutionary algorithms

Illustration from *Metaheuristics - From design to implementation*

Main questions:

- What representations?
  - ✓ sequential structure (bit or float)
  - ✓ finite automaton
  - ✓ tree structure
- What selection mechanism?
  - ✓ roulette wheel
  - ✓ tournaments
- What reproduction operators?
  - ✓ mutation (unary operator)
  - ✓ crossover (binary operator)
- What replacement strategy?
- What stopping criteria?

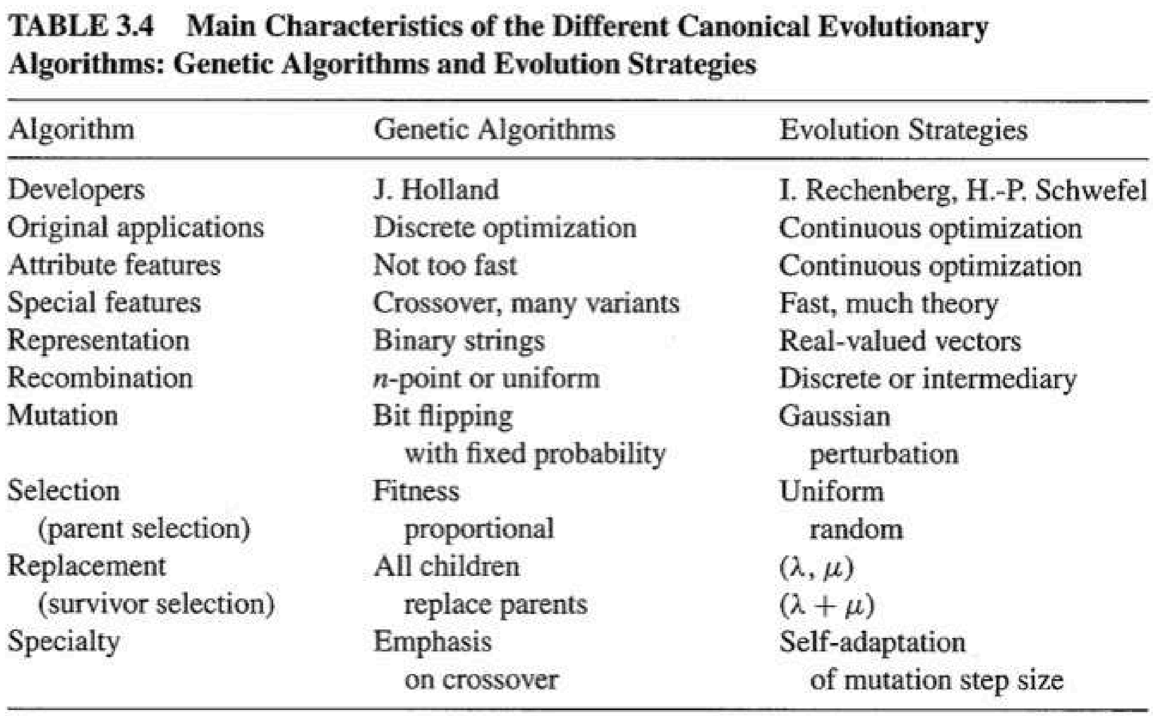


TABLE 3.4 Main Characteristics of the Different Canonical Evolutionary Algorithms: Genetic Algorithms and Evolution Strategies

Algorithm	Genetic Algorithms	Evolution Strategies
Developers	J. Holland	I. Rechenberg, H.-P. Schwefel
Original applications	Discrete optimization	Continuous optimization
Attribute features	Not too fast	Continuous optimization
Special features	Crossover, many variants	Fast, much theory
Representation	Binary strings	Real-valued vectors
Recombination	n-point or uniform	Discrete or intermediary
Mutation	Bit flipping with fixed probability	Gaussian perturbation
Selection (parent selection)	Fitness proportional	Uniform random
Replacement (survivor selection)	All children replace parents	(λ, μ) $(\lambda + \mu)$
Specialty	Emphasis on crossover	Self-adaptation of mutation step size

Table from *Metaheuristics - From design to implementation*

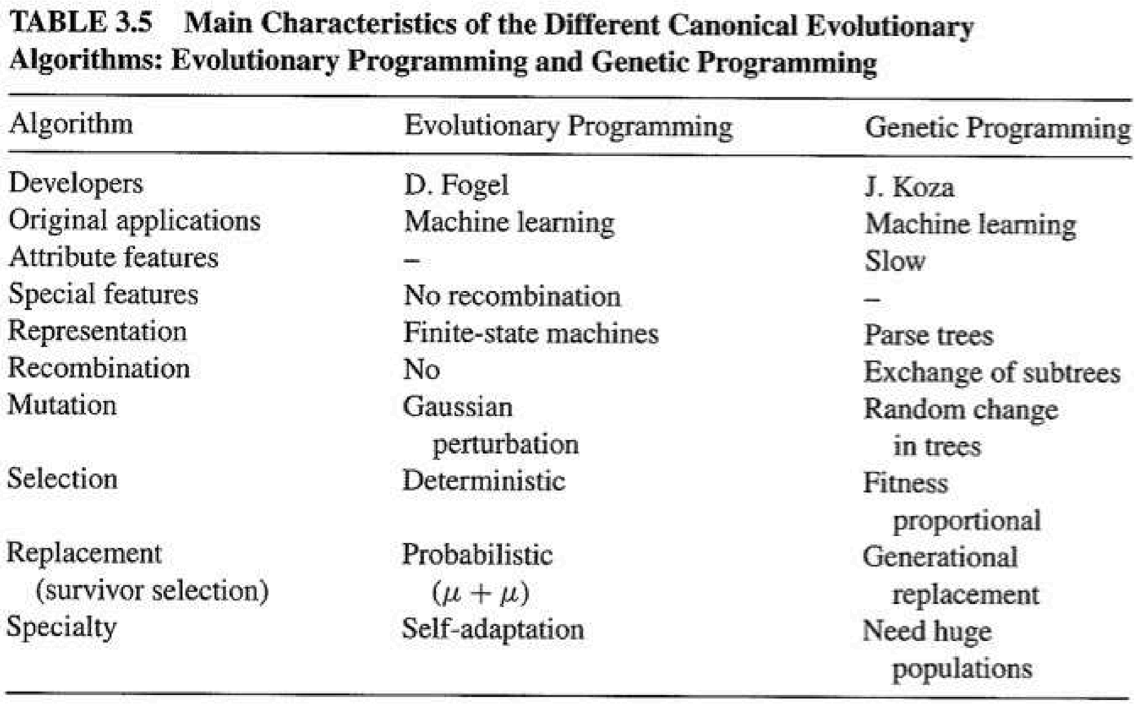


TABLE 3.5 Main Characteristics of the Different Canonical Evolutionary Algorithms: Evolutionary Programming and Genetic Programming

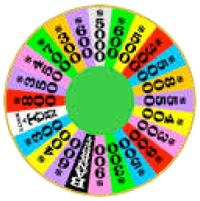
Algorithm	Evolutionary Programming	Genetic Programming
Developers	D. Fogel	J. Koza
Original applications	Machine learning	Machine learning
Attribute features	–	Slow
Special features	No recombination	–
Representation	Finite-state machines	Parse trees
Recombination	No	Exchange of subtrees
Mutation	Gaussian perturbation	Random change in trees
Selection	Deterministic	Fitness proportional
Replacement (survivor selection)	Probabilistic $(\mu + \mu)$	Generational replacement
Specialty	Self-adaptation	Need huge populations

Table from *Metaheuristics - From design to implementation*

Genetic algorithms

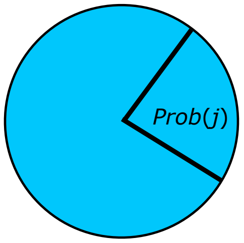
- Representations
  - ✓ binary strings
  - ✓ sequence of integers / permutations
  - ✓ vectors of floats
- Reproduction using crossover operations
- Mutations to promote diversity
- Generational replacement

Selection

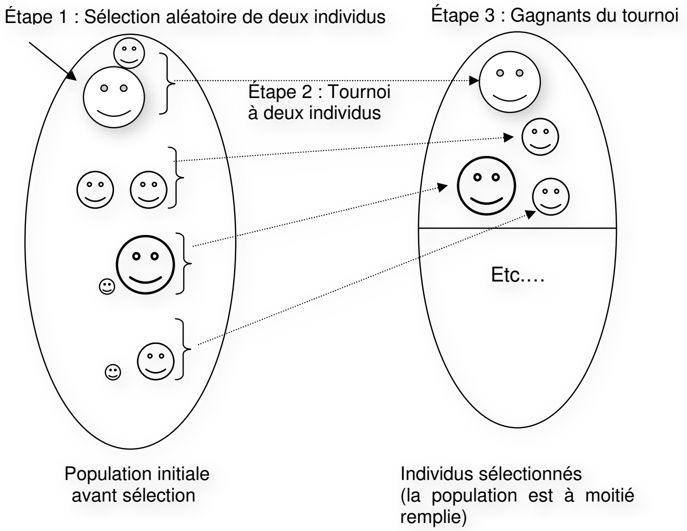


wheel of fortune

$$Prob\,(j) = \frac{Fitness(j)}{\sum_{j=1}^{Jmax} Fitness(j)}$$



tournaments



Étape 1 : Sélection aléatoire de deux individus

Étape 2 : Tournoi à deux individus

Étape 3 : Gagnants du tournoi

Etc.…

Population initiale avant sélection

Individus sélectionnés (la population est à moitié remplie)

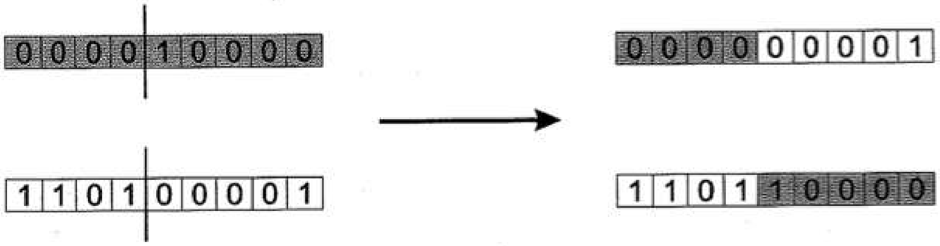


Fig. 3.6. One-point crossover

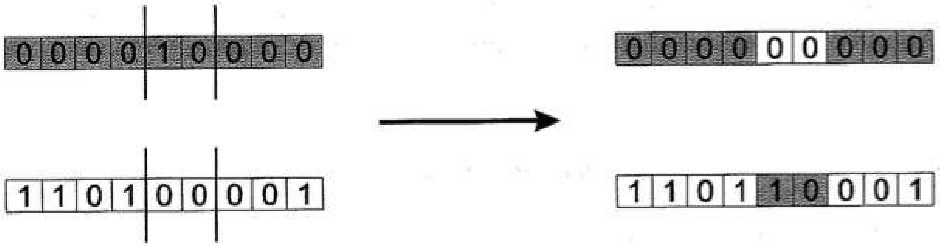


Fig. 3.7. n-point crossover: $n = 2$

Illustration from *Introduction to Evolutionary Computing*

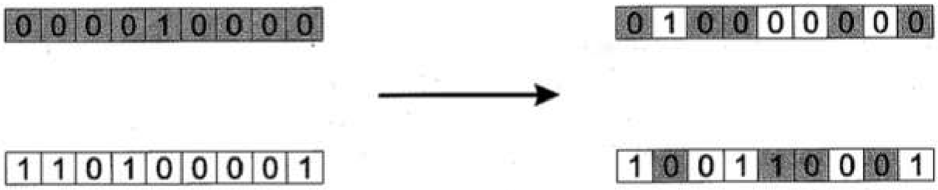


Fig. 3.8. Uniform crossover. In this example the array [0.35, 0.62, 0.18, 0.42, 0.83, 0.76, 0.39, 0.51, 0.36] of random numbers drawn uniformly from [0,1) was used to decide inheritance

Illustration from *Introduction to Evolutionary Computing*

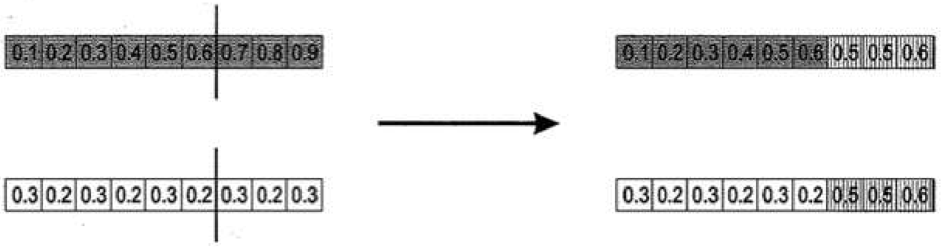


Fig. 3.9. Simple arithmetic recombination: $k = 6, \alpha = 1/2$

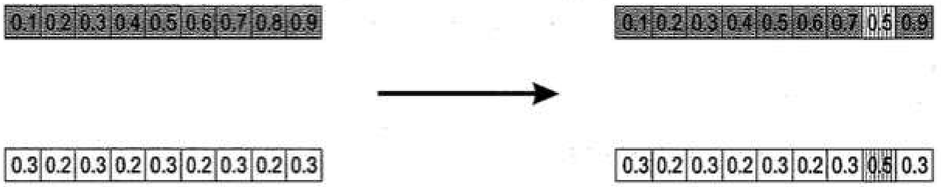


Fig. 3.10. Single arithmetic recombination: $k = 8, \alpha = 1/2$

Illustration from *Introduction to Evolutionary Computing*

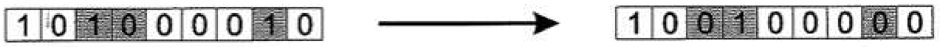


Fig. 3.1. Bitwise mutation for binary encodings

Illustration from *Introduction to Evolutionary Computing*

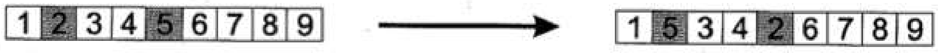


Fig. 3.2. Swap mutation

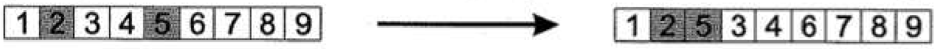


Fig. 3.3. Insert mutation

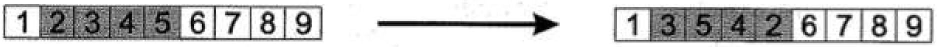


Fig. 3.4. Scramble mutation

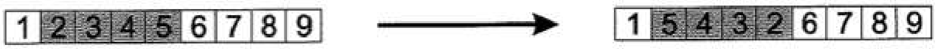


Fig. 3.5. Inversion mutation

Illustration from *Introduction to Evolutionary Computing*

Evolutionary Strategies

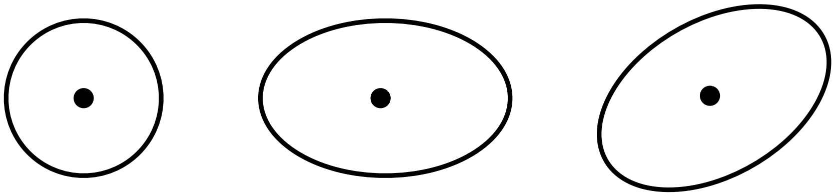
- Representation: vector of floats
- Crossover rarely used
- Continuous optimization using self-adaptation *Gaussian mutations*
- Special (μ,λ) or (μ+λ) replacement strategy
  - ✓ μ is the parents size
  - ✓ λ is the offsprings size

Basic ES template

```
Initialize a population of μ individuals;
Evaluate the μ individuals;
Repeat
  - Generate λ offsprings from the μ parents;
  - Evaluate the λ offsprings;
  - Replace the population with μ individuals
    taken from parents and offsprings;
Until stopping criteria satisfied
Output best individual or population found
```

Gaussian mutations

- Consists in a random perturbation of the underlying vector



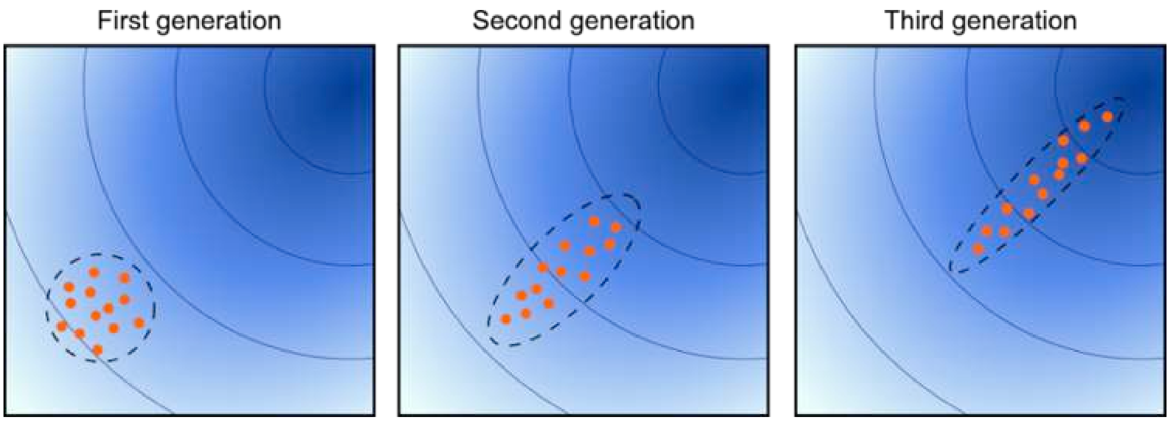
uncorrelated single σ

uncorrelated diagonal Σ

correlated full Σ

- Self-adapting correlation matrix

Covariance Matrix Adaptation (CMA-ES)



1 individual = vector **x** + matrix **Σ**

Evolutionary programming

- Representation: finite-state automaton
  - ✓ binary or float
- Crossover rarely used
- Mutations
  - ✓ bit flip or Gaussian
- (µ+µ) replacement strategy
  - ✓ µ is the parents size
  - ✓ µ is the offsprings size

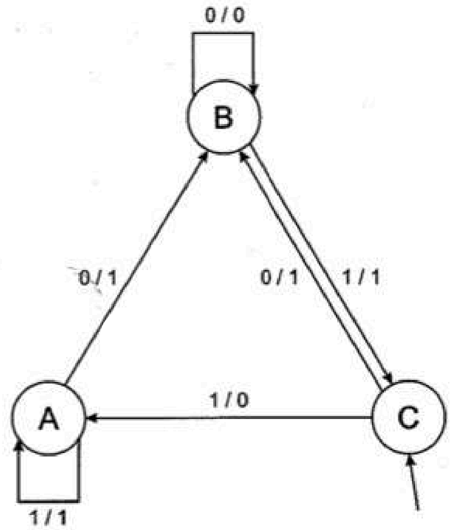


Fig. 5.2. Finite state machine as a predictor. The initial state of this FSM is C, and the given input string is 011101. The FSM's response is the output string 110111. On this string, its prediction performance is 60% (inputs 2, 3, and 6 correctly predicted)

Illustration from *Introduction to Evolutionary Computing*

Mutations operators

- Changing an output symbol
- Changing a state transition
- Adding a new state
- Deleting a state
- Changing the initial state

Genetic programming

- Representation: parse tree
- Recombinations and mutations operate on subtrees
- Generational replacement

$2\pi + ((x+3) - y/(5+1))$ $(x \wedge true) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$

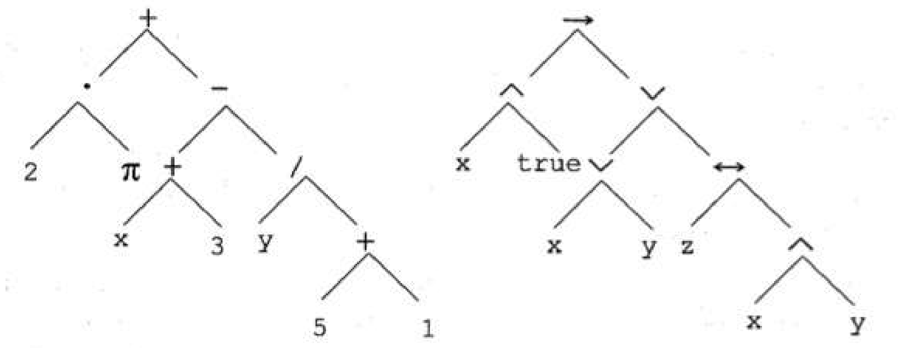


Fig. 6.2. Parse trees belonging to Eqs. (6.2) (*left*) and (6.3) (*right*)

Illustration from *Introduction to Evolutionary Computing*

```
i = 1;
while ( i < 20 )
{
  i = i+1;
}
```

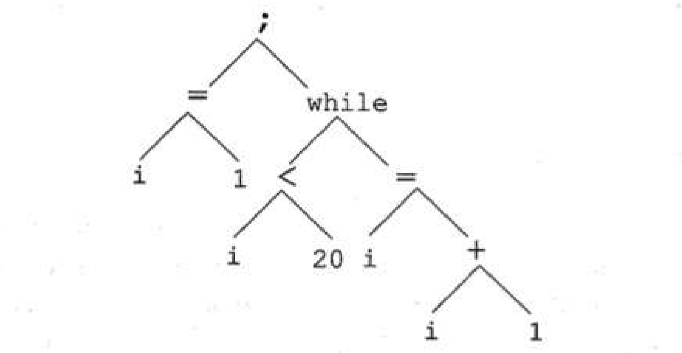


Fig. 6.3. Parse tree belonging to the above program

Illustration from *Introduction to Evolutionary Computing*

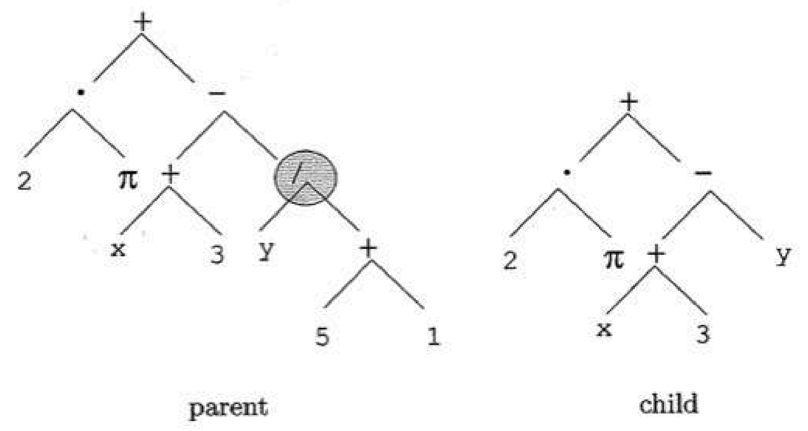


Fig. 6.5. GP mutation illustrated: the node designated by a *circle* in the tree *on the left* is selected for mutation. The subtree staring at that node is replaced by a randomly generated tree, which is a leaf here

Illustration from *Introduction to Evolutionary Computing*

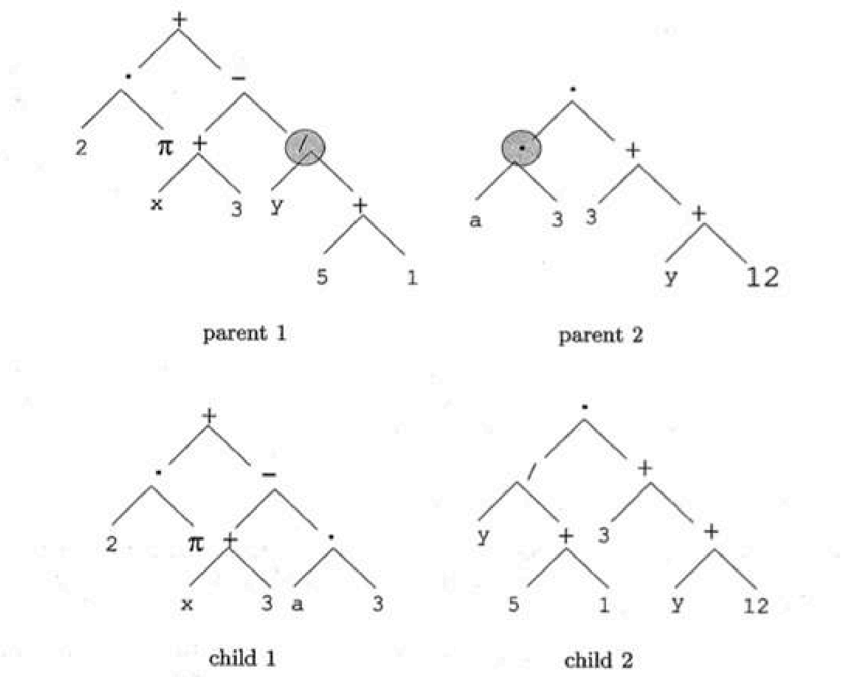


Fig. 6.6. GP crossover illustrated: the nodes designated by a *circle* in the parent trees are selected to serve as crossover points. The subtrees staring at those nodes are swapped, resulting in two new trees, which are the children

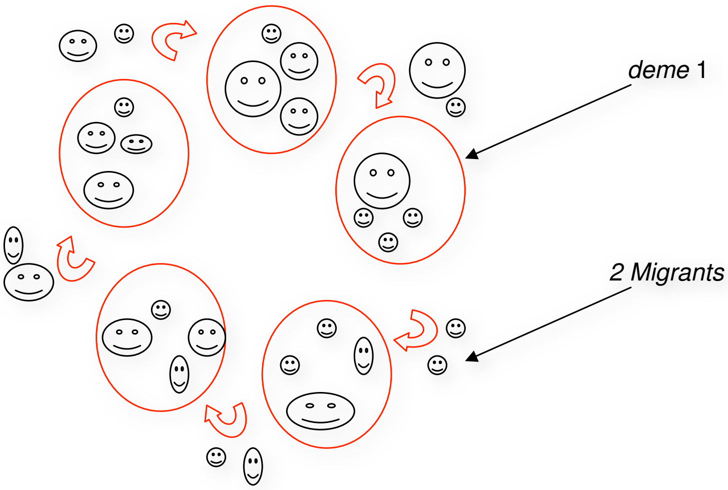
Illustration from *Introduction to Evolutionary Computing*

Primitive operations

- Tree branches correspond to elementary operations that can be applied on data to solve the problem at hand
  - ✓ the user must specify the set of applicable primitives
- Tree leaves (terminals) are terminal symbols, that is input variables, constants, or random values
- Trees are generated by randomly picking primitives and terminals

Island model

- Demes are sub-population that evolve in isolation
- Periodically, some travellers migrate from one deme to the other



Coevolution

- Two or more species that either compete or cooperate through evolution

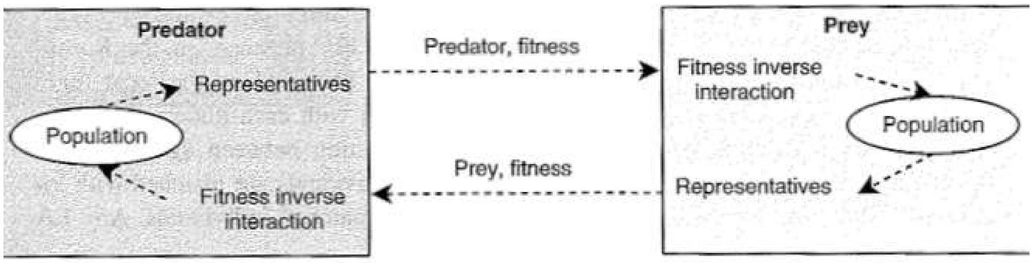


FIGURE 3.26 Competitive coevolutionary algorithms based on the predator–prey model.

Illustration from *Metaheuristics - From design to implementation*

- The solution is the assembly of the different species
- individuals from the different species are randomly matched

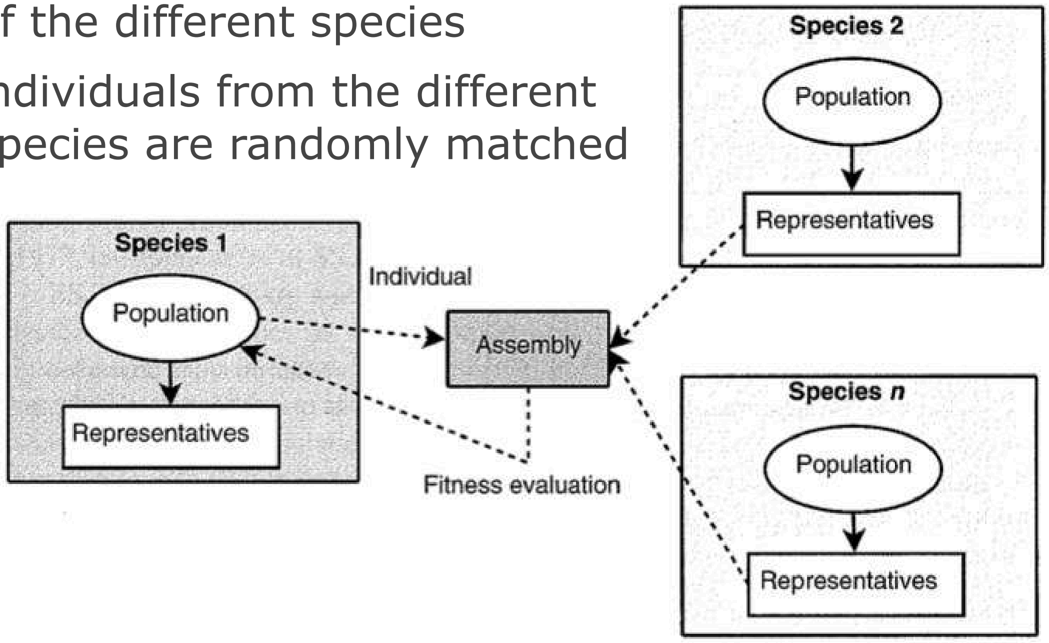


FIGURE 3.27 A cooperative coevolutionary algorithm.

Illustration from *Metaheuristics - From design to implementation*

Exploration vs exploitation

- Evolutionary algorithms are good at ***exploring*** the solution space of the problem
  - ✓ because of their parallel nature
- Local search method are good at ***exploiting*** local neighbourhoods
  - ✓ but they get stuck in local optima

Hybrid methods

- Combining local search to EAs
- Memetic algorithms
  - ✓ adding a developmental learning phase within the evolutionary cycle

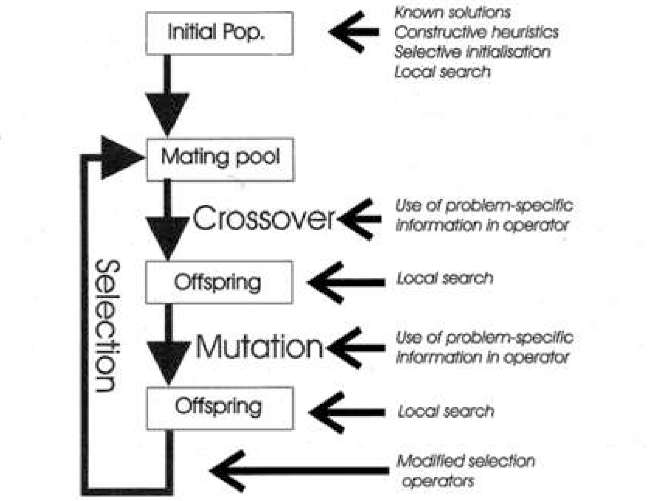


Fig. 10.3. Possible places to incorporate knowledge or other operators within the evolutionary cycle

Illustration from *Introduction to Evolutionary Computing*

Part I: ***fundamentals***

- Evolutionary computations for artificial intelligence?
- Flavours of evolutionary Algorithms
- Multiobjective optimization
- Parallelism

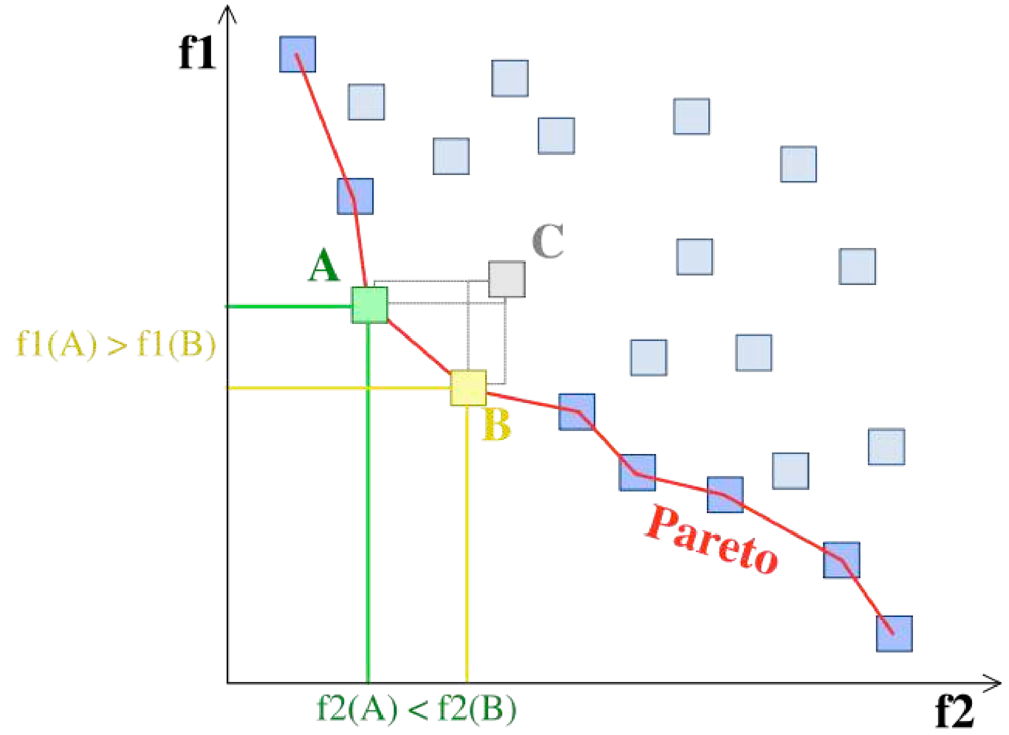
Multiobjective optimization

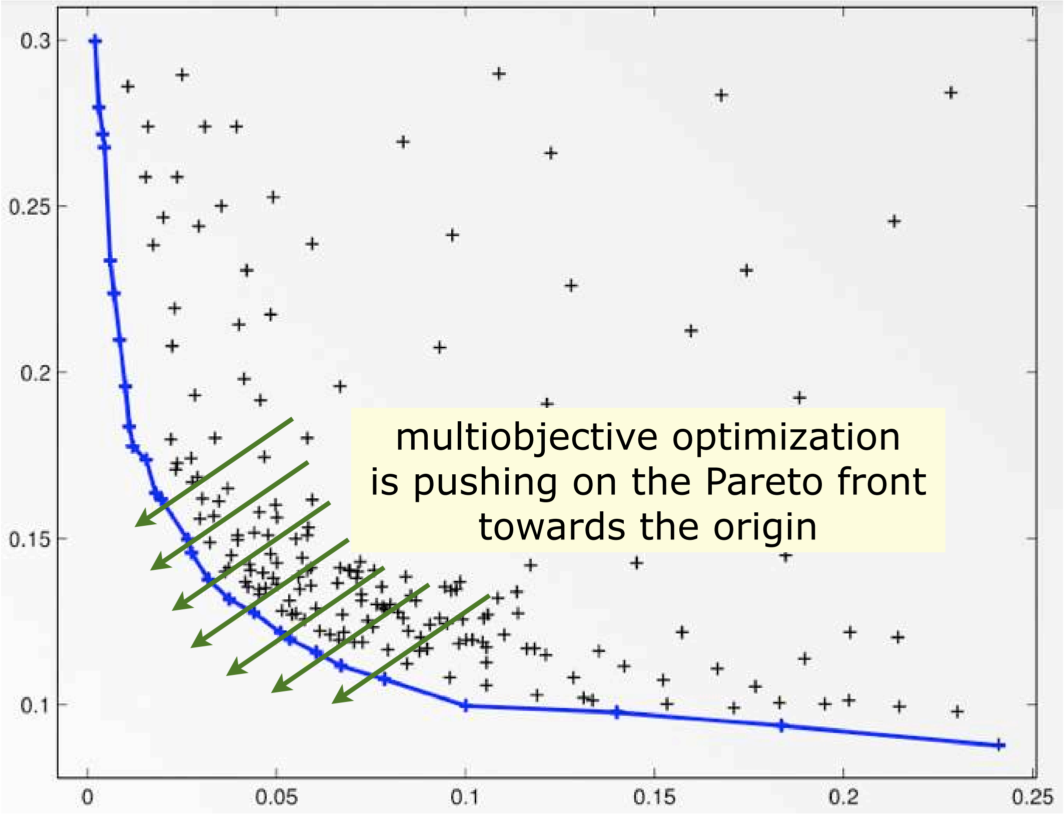
- Multicriteria decision making
  - ✓ e.g. cost vs performance
- Pareto dominance
- Pareto front
- NSGA-II

Pareto dominance

- A vector of objectives **u**=(u_1,...,u_n) is said to dominate **v**=(v_1,...,v_n) *iff* no component of **v** is better then those of **u** and at least one component of **u** is better than the corresponding component of **v**

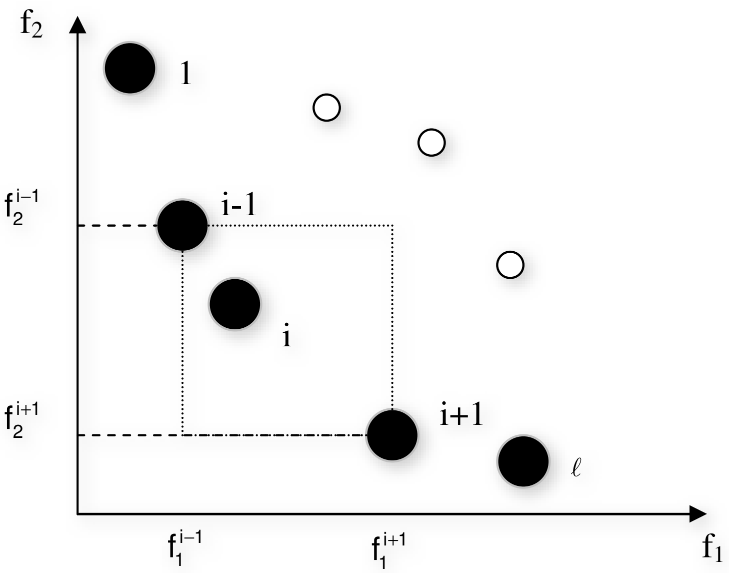
$$\forall i \in \{1, \ldots, n\} : u_i \leq v_i \ \wedge \ \exists i \in \{1, \ldots, n\} : u_i < v_i$$



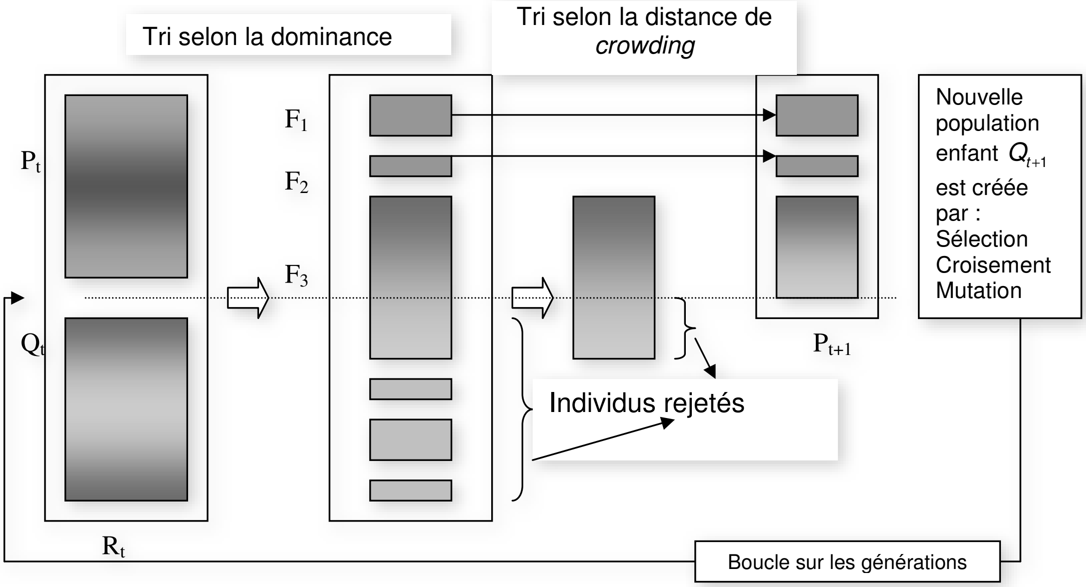


multiobjective optimization
is pushing on the Pareto front
towards the origin

Crowding distance



Non-dominated sorting (NSGA-II)

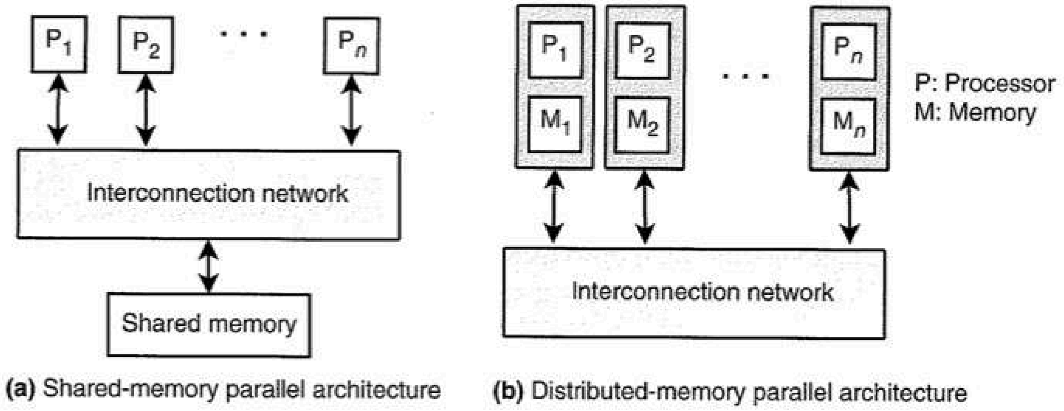


Tri selon la dominance
Tri selon la distance de *crowding*
P_t
Q_t
R_t
F_1
F_2
F_3
P_{t+1}
Individus rejetés
Nouvelle population enfant Q_{t+1} est créée par : Sélection Croisement Mutation
Boucle sur les générations

Part I: ***fundamentals***

- Evolutionary computations for artificial intelligence?
- Flavours of evolutionary Algorithms
- Multiobjective optimization
- Parallelism

Shared vs distributed memory



(a) Shared-memory parallel architecture **(b)** Distributed-memory parallel architecture

FIGURE 6.16 Shared memory versus distributed memory architectures.

Illustration from *Metaheuristics - From design to implementation*

Non-Uniform Memory Access (NUMA)

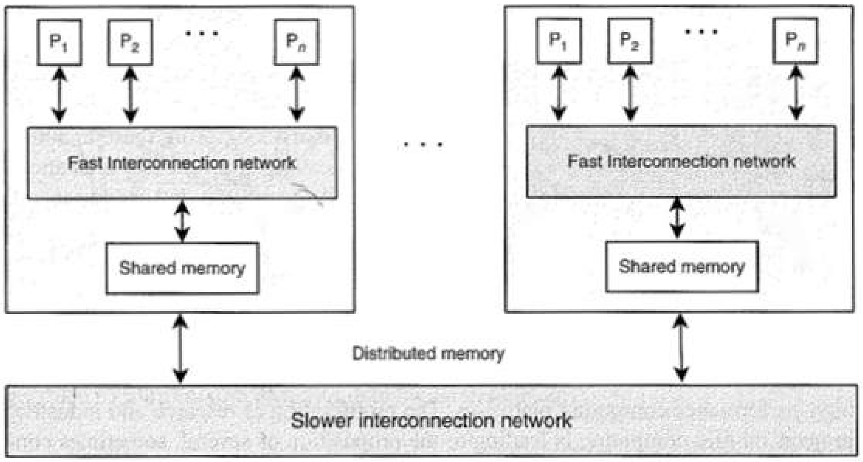


FIGURE 6.18 ccNUMA architectures combining shared and distributed memory architectures represent actually the most powerful machines (e.g., Earth Simulator, Cray X1, SGI Origin2000).

Off-the-shelf processors today are of the NUMA type!

For example, the new Intel Nehalem architecture (*iCore* 7)

Illustration from *Metaheuristics - From design to implementation*

Multithreading

- Multiple threads of execution within a single process
- All threads share the same memory space
- Requires synchronization locks to protect shared variables

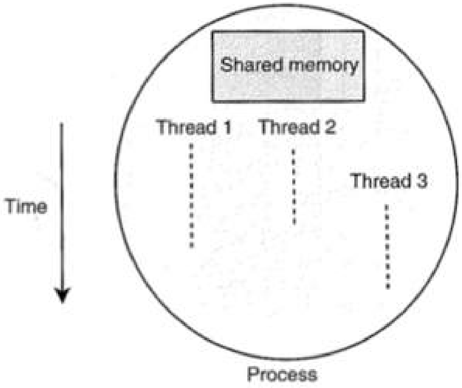
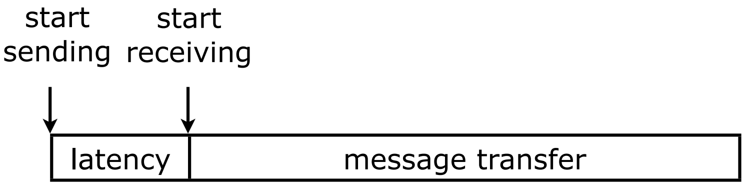


Illustration from *Metaheuristics - From design to implementation*

Memory wall

- High bandwidth
  - ✓ to quickly transfer large messages
- Low latency
  - ✓ to be able to send many short messages



Memory access balance

- Some architecture deliberately choose slower CPUs to better balance access time between shared and distributed memory
  - ✓ for example the Blue Gene architecture from IBM

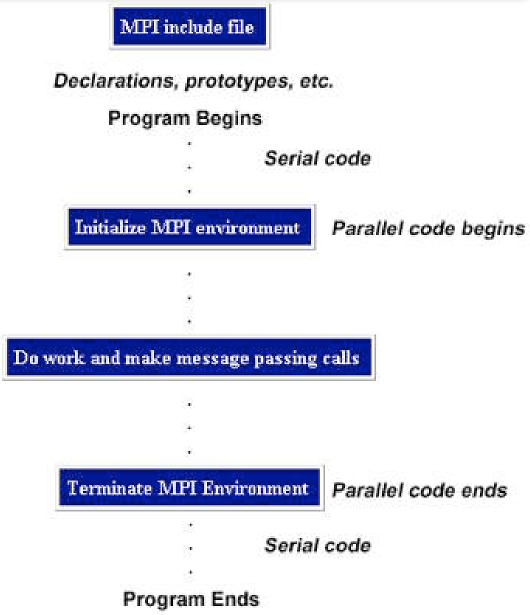
CPUs consume less power;
so they can put more inside
a cabinet!

Message Passing Interface (MPI)

- Standard specification for message passing libraries
  - ✓ practical
  - ✓ portable
  - ✓ efficient
  - ✓ flexible
- Interfaces in C, C++, and Fortran
  - ✓ also some support for other languages

Program structure

- The same program runs on many process
- Each process has a unique ID called the MPI rank
- Messages can be send or received by ranks or by group of ranks



MPI include file

Declarations, prototypes, etc.

Program Begins

Serial code

Initialize MPI environment

Parallel code begins

Do work and make message passing calls

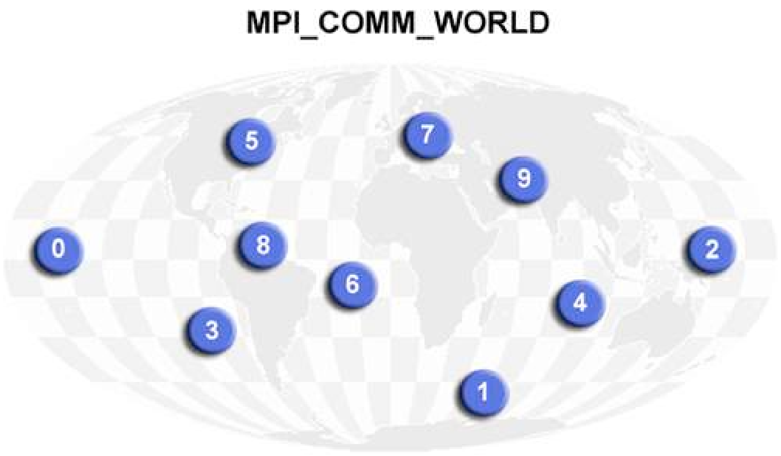
Terminate MPI Environment

Parallel code ends

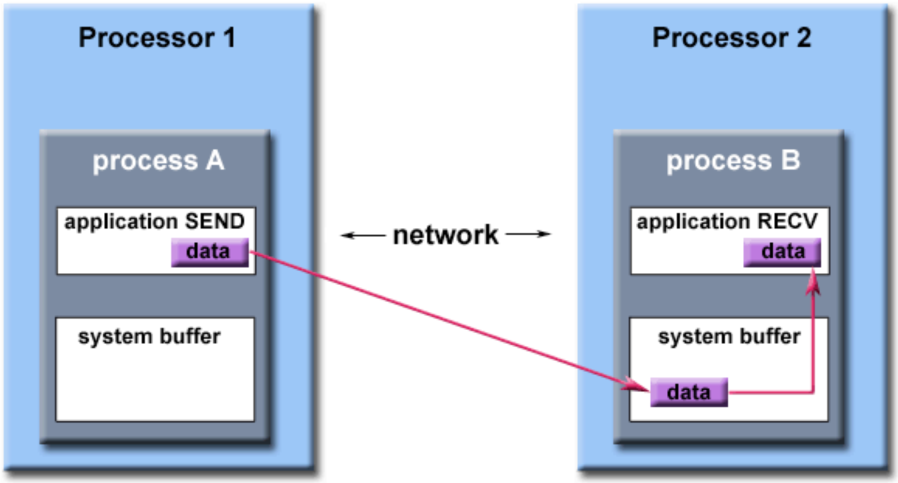
Serial code

Program Ends

Communicators



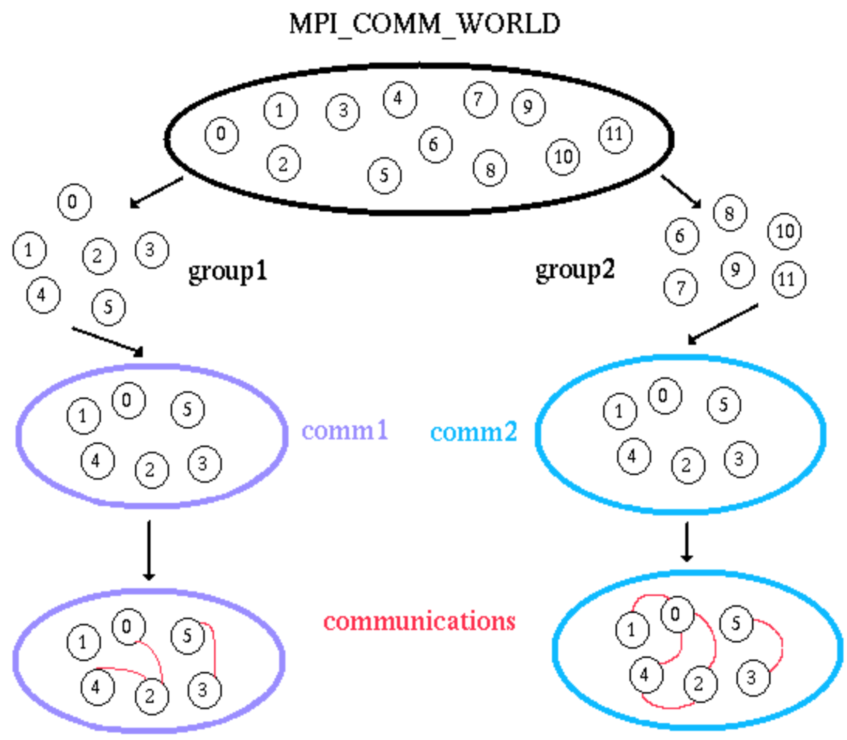
Buffering



Path of a message buffered at the receiving process

Blocking vs non blocking

- Blocking:
  - ✓ a send will only return when it is safe to reuse the message buffer
  - ✓ a receive only returns after data has arrived and is ready to use
- Non blocking:
  - ✓ send and receive will return almost immediately
  - ✓ if no data is available, receive returns with fail status
  - ✓ user cannot predict when operations will be complete



0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

Conclusion

- EAs are both *powerful* and diverse
- But they require much computational effort to *solve* real world problems
- However, they are also *embarrassingly* parallel!
- Great *speedups* are achievable using *parallel architectures*

Part II: ***tools***

- Hardware: colossus
  - ✓ CLUMEQ
- Software
  - ✓ requirements
  - ✓ survey
- Open BEAGLE

Hardware requirements

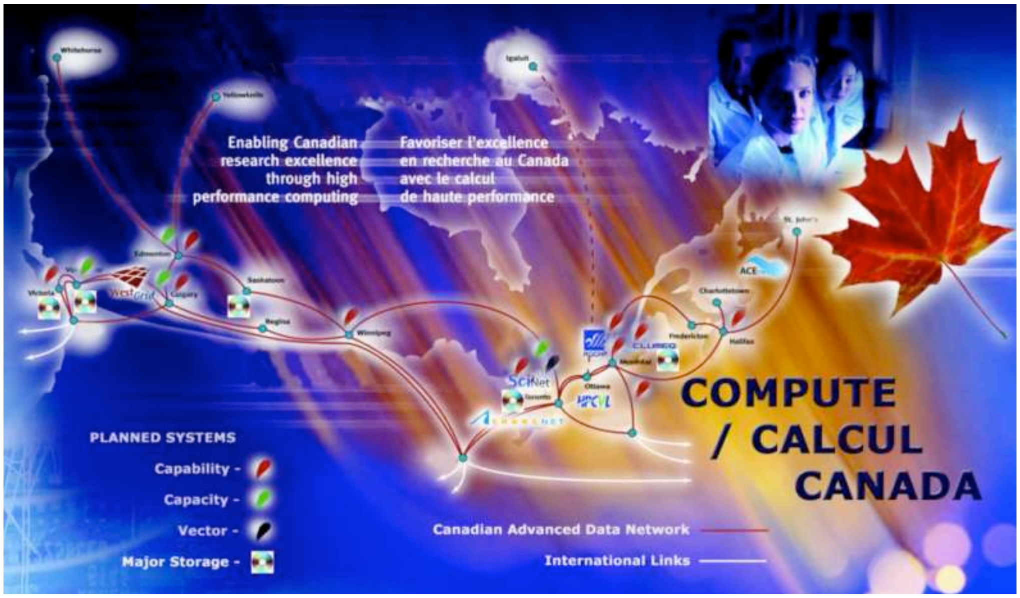
- EAs are compute intensive,
  - ✓ but embarrassingly parallel!
- Real world problems are hard,
  - ✓ because solution spaces are vast
  - ✓ and objectives are many
- Clock frequencies are not expected to increase,
  - ✓ but processors are now multicore
- Tools should be designed from the start to efficiently exploit parallelism
  - ✓ I wish everything could be "*automagic*"!

CLUMEQ

- Consortium of 11 universities in the province of Québec, Canada

McGill
UNIVERSITÉ LAVAL
UQÀM
UQAR
Université du Québec
UQAC Université du Québec à Chicoutimi
Université du Québec en Outaouais
Université du Québec à Trois-Rivières
Université du Québec en Abitibi-Témiscamingue
École nationale d'administration publique
Université du Québec Institut national de la recherche scientifique
Université du Québec École de technologie supérieure

Compute Canada
The national HPC platform



Québec site

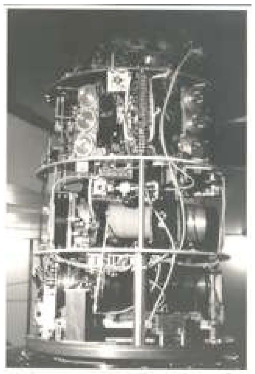
- Silo of a decommissioned Van de Graaff particule accelerator
- Recycled as a cooling enclosure for a supercomputer

exterior view
(circa 1965)

control room

computer room

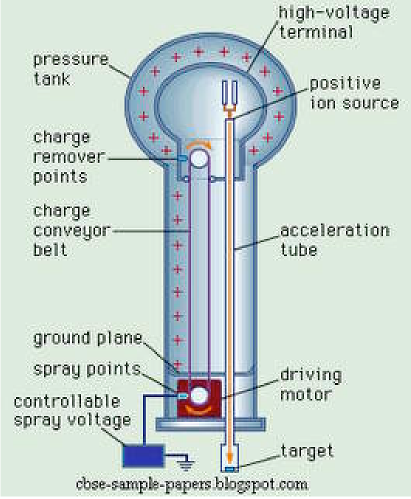
accelerator



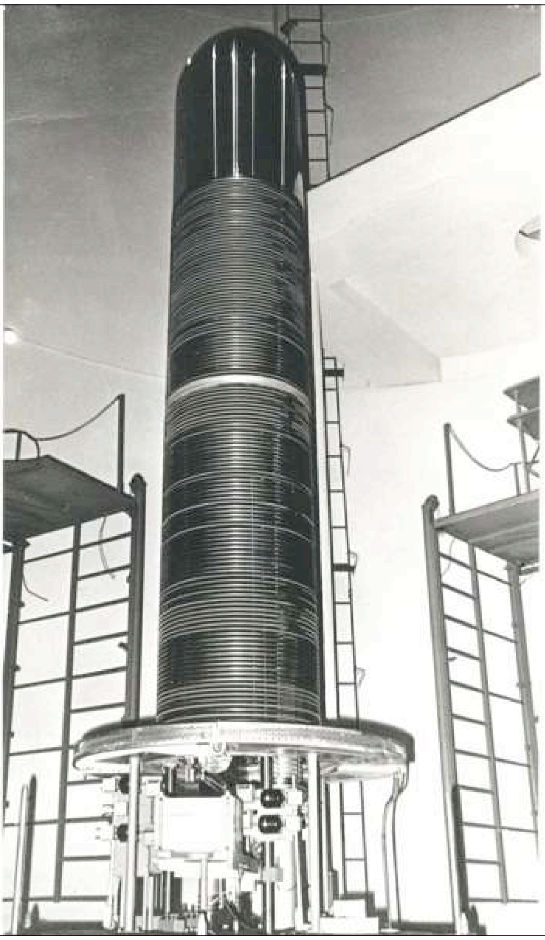
upper part

target room

Van de Graaff particle accelerator

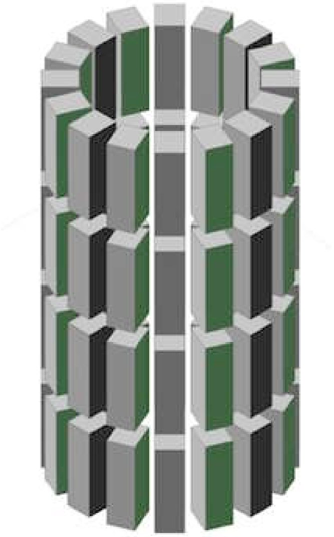


cbse-sample-papers.blogspot.com



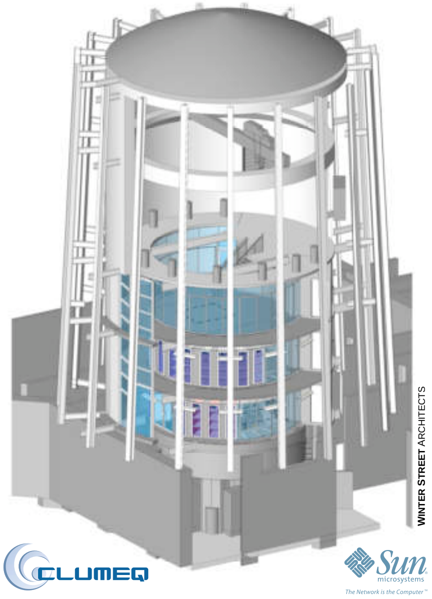
Concept

- Unique in the world
- Compute racks arranged in a cylindrical topology
- Inner hot-air core
- Outer cold air ring-shape plenum
  - ✓ low air velocity, because of high cross-section
  - ✓ no corners to produce turbulence

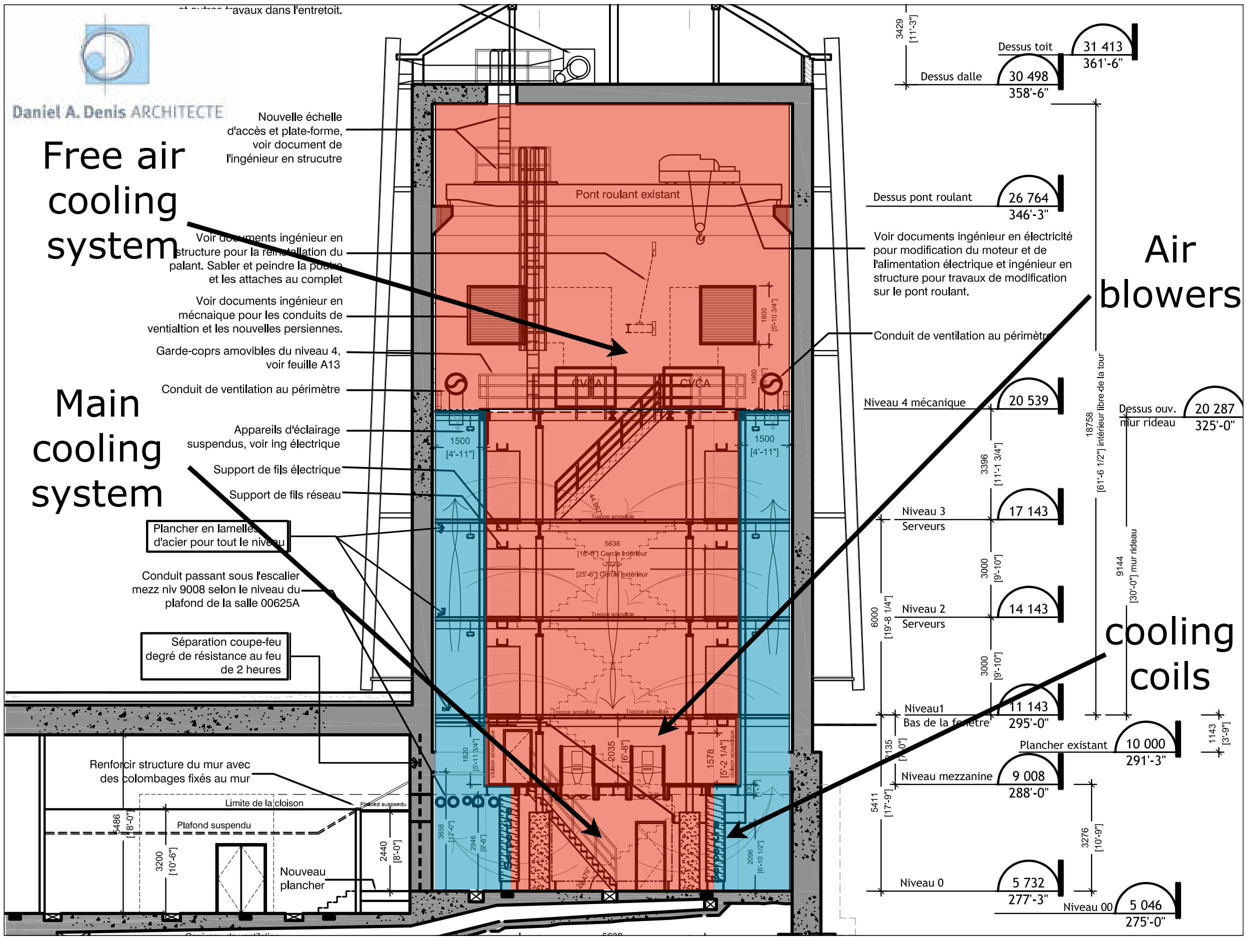


Main specifications

- up to 56 standard size racks on 3 levels
- up to 1.2 megawatts of power & cooling
- up to 132,500 CFM of blowing power
- very efficient cooling system
  - ✓ capable of recycling heat
  - ✓ capable of free air cooling



CLUMEQ
WINTER STREET ARCHITECTS
Sun microsystems

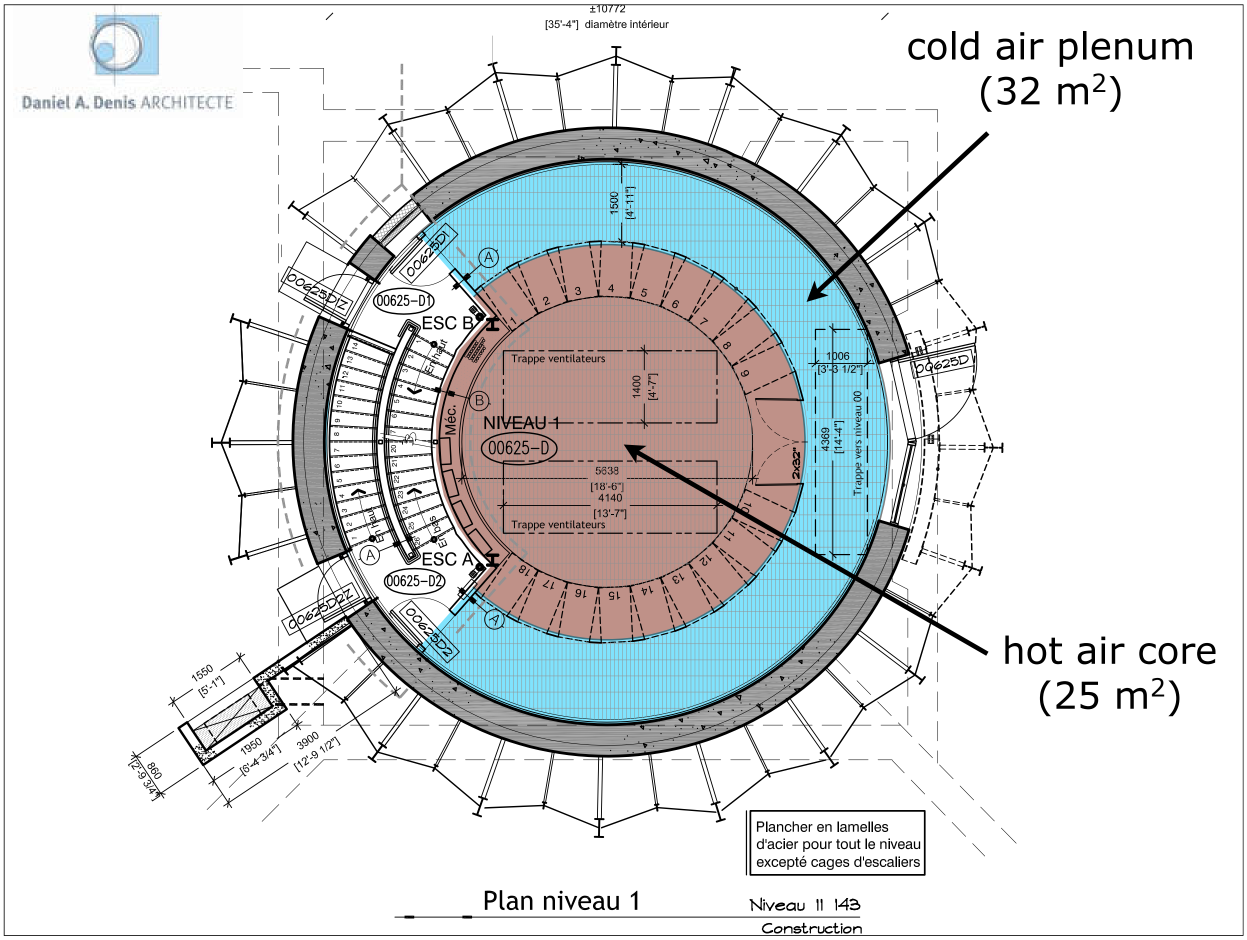

et autres travaux dans l'entretoit.
Daniel A. Denis ARCHITECTE
Nouvelle échelle d'accès et plate-forme, voir document de l'ingénieur en strucutre
Voir documents ingénieur en structure pour la reinstallation du palant. Sabler et peindre la poutre et les attaches au complet
Voir documents ingénieur en mécnaique pour les conduits de ventialtion et les nouvelles persiennes.
Garde-coprs amovibles du niveau 4, voir feuille A13
Conduit de ventilation au périmètre
Appareils d'éclairage suspendus, voir ing électrique
Support de fils électrique
Support de fils réseau
Plancher en lamelles d'acier pour tout le niveau
Conduit passant sous l'escalier mezz niv 9008 selon le niveau du plafond de la salle 00625A
Séparation coupe-feu degré de résistance au feu de 2 heures
Renforcir structure du mur avec des colombages fixés au mur
Limite de la cloison
Plafond suspendu
Nouveau plancher
Pont roulant existant
CVCA
CVCA
1500 [4'-11"]
1500 [4'-11"]
3429 [11'-3"]
Dessus toit 31 413 361'-6"
Dessus dalle 30 498 358'-6"
Dessus pont roulant 26 764 346'-3"
Voir documents ingénieur en électricité pour modification du moteur et de l'alimentation électrique et ingénieur en structure pour travaux de modification sur le pont roulant.
Conduit de ventilation au périmètre
Niveau 4 mécanique 20 539
18758 [61'-6 1/2"] intérieur libre de la tour
Dessus ouv. mur rideau 20 287 325'-0"
3396 [11'-1 3/4"]
Niveau 3 Serveurs 17 143
3000 [9'-10"]
9144 [30'-0"] mur rideau
6000 [19'-8 1/4"]
Niveau 2 Serveurs 14 143
3000 [9'-10"]
Niveau1 Bas de la fenetre 11 143 295'-0"
Plancher existant 10 000 291'-3"
1143 [3'-9"]
Niveau mezzanine 9 008 288'-0"
5411 [17'-9"]
3276 [10'-9"]
Niveau 0 5 732 277'-3"
Niveau 00 5 046 275'-0"
5486 [18'-0"]
3200 [10'-6"]
2440 [8'-0"]

Free air cooling system

Main cooling system

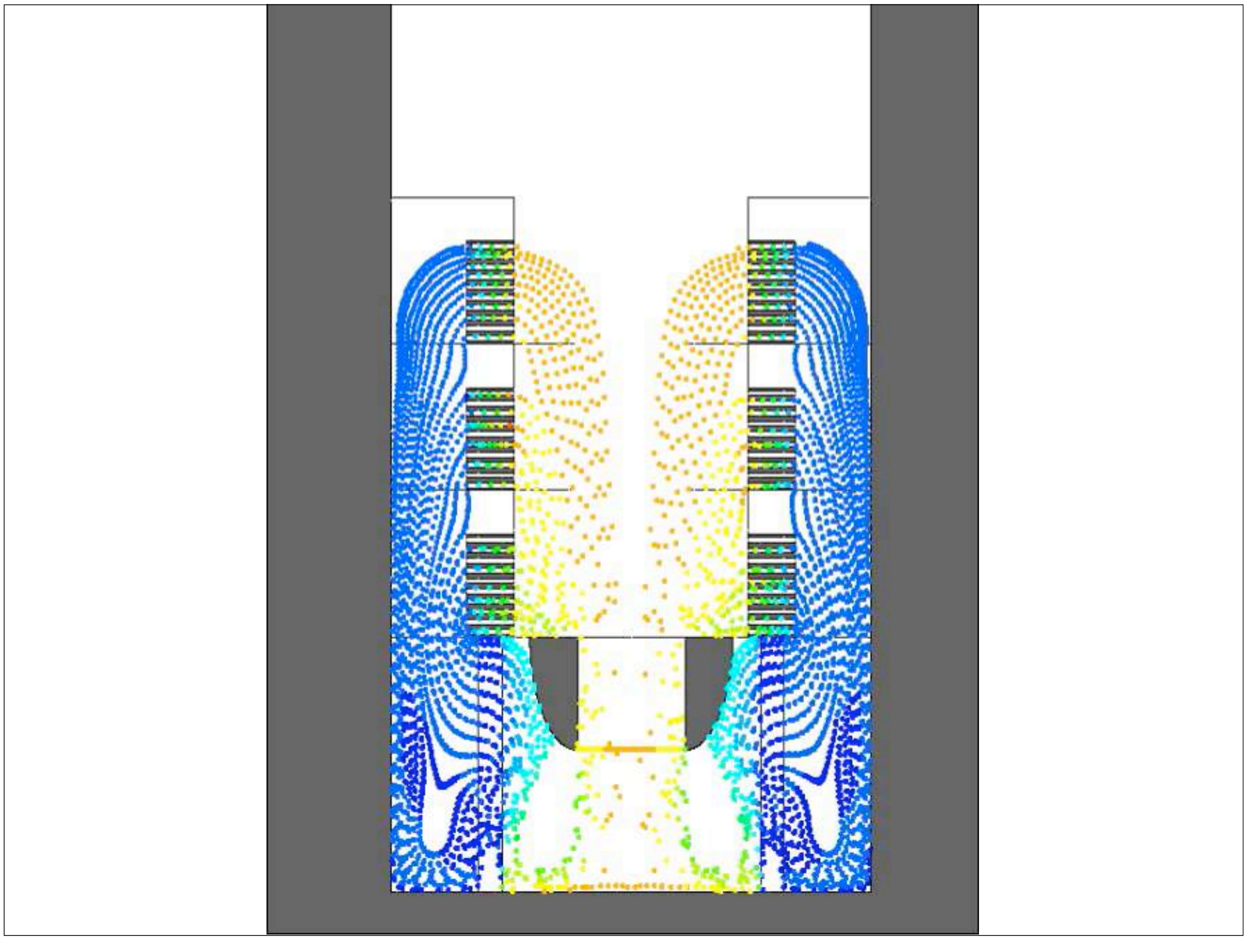
Air blowers

cooling coils



cold air plenum
(32 m^2)

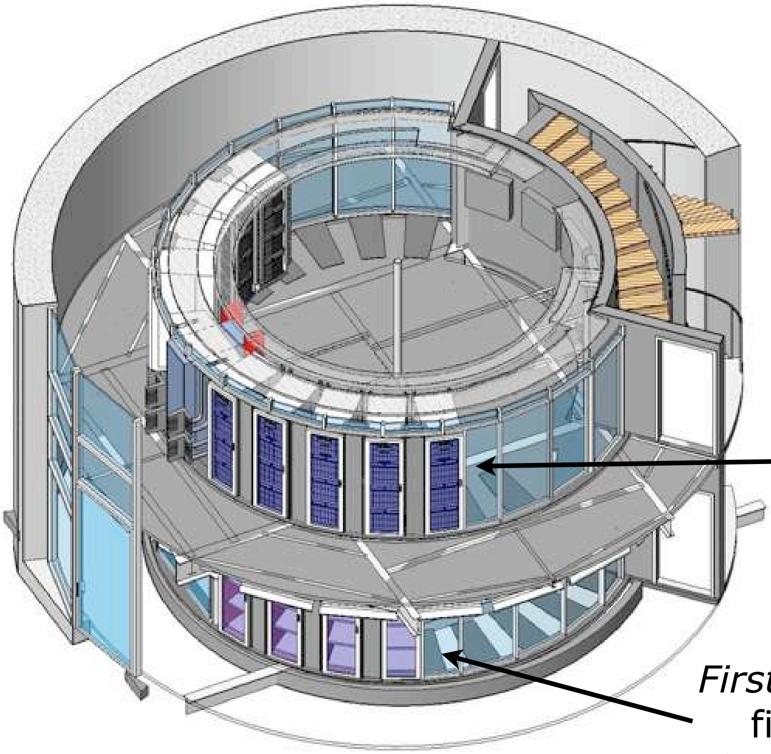
hot air core
(25 m^2)



Colossus cluster

- Sun constellation system
  - ✓ 10 fully loaded Sun Blade 6048, with X6275 modules (double Nehalem EP blade, 2.8GHz, 24GB of RAM)
  - ✓ full-bisection IB-QDR interconnect (2xM9 switches)
  - ✓ 1 PB of Lustre storage in a high availability configuration, using 2 MDS and 9x2 OSS
  - ✓ Sun J4400 storage arrays

- 40 infrastructure nodes
- 960 compute nodes
- 1920 CPU sockets (Nehalem-EP 2.8GHz)
- 7680 processor cores
- about 23 TB of RAM
- 500 TB of disk (will be upgraded to 1 PB)
- Full bisection 40 Gb/sec networking between compute nodes (no bottlenecks)
- 10 Gb/sec Ethernet to the university backbone



Racks aligned in a circle around a central hot core; outside ring is a cold air plenum

Second floor contains all compute racks + core networking switches

First floor contains file system & infrastructure nodes

Sun Blade 6048 Shelf with x6275

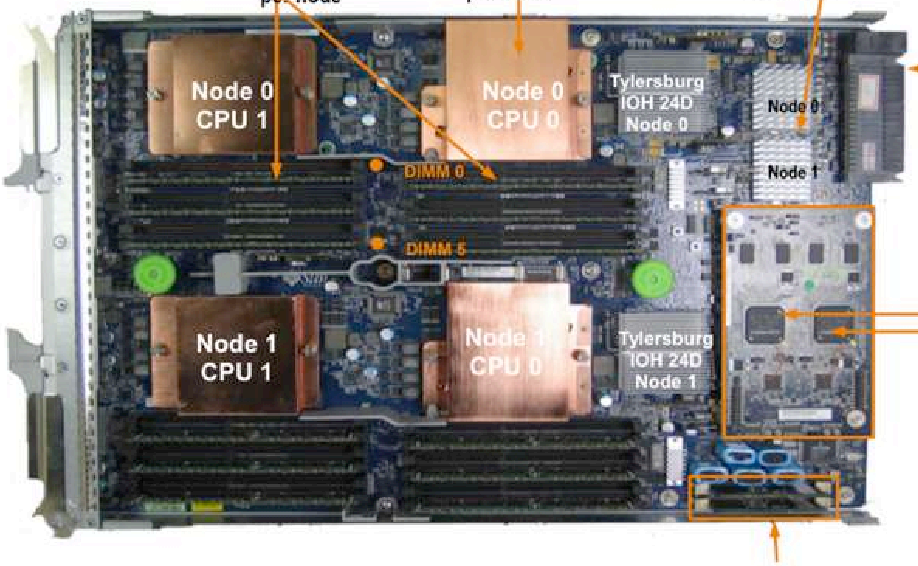
x6275

12 DDR3 slots. Up to 48GB of RAM per node

Two Intel Nehalem-EP quad-core processors per node.

Mellanox MT25408 Infiniband

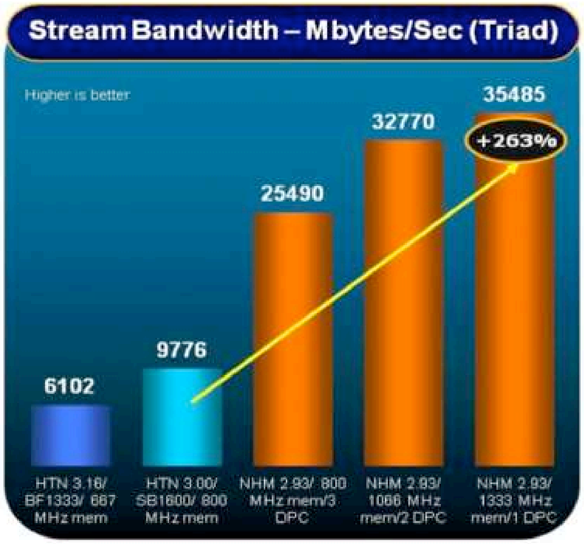
Four PCIe interfaces, four SAS links and two 1GbEth ports



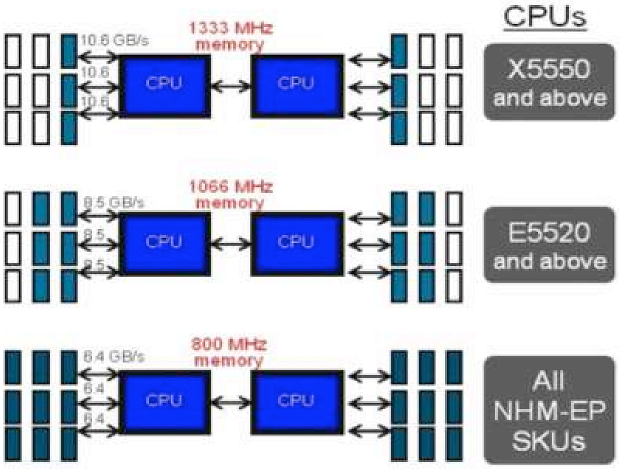
ASPEED AST2100

2 Flash Modules (1 for each node)

Nehalem-EP Memory Flexibility

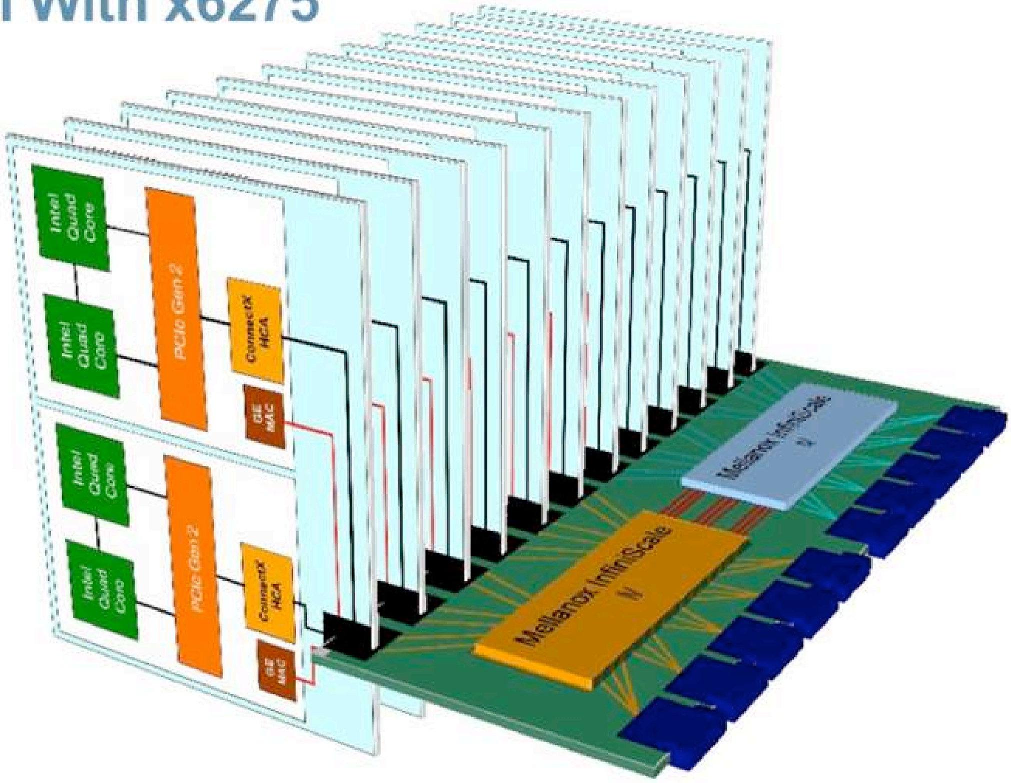


(DPC – Dimms Per Channel)



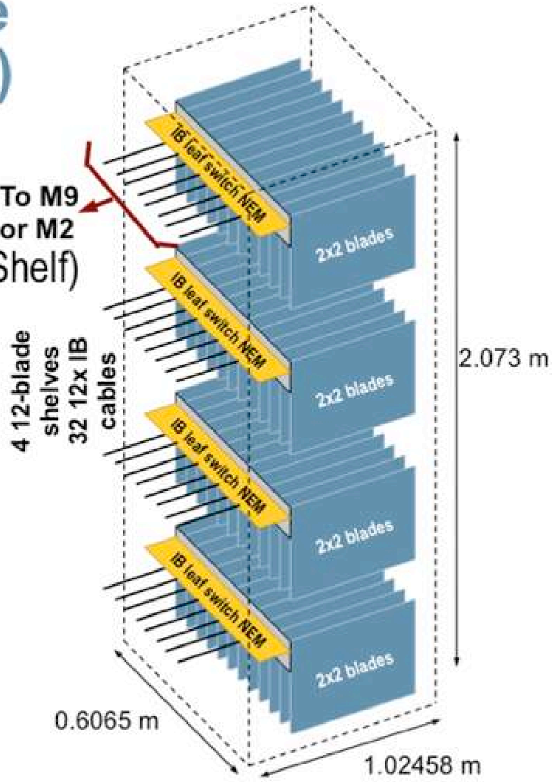
NHM-EP memory Bandwidth for different configuration						
Memory speed	800 MHz			1066 MHz		1333 MHz
	1 DPC	2 DPC	3 DPC	1 DPC	2 DPC	1 DPC
Stream Triad	26757	25539	25490	32629	32770	35485

QNEM With x6275



Constellation 48-blade Rack (Sun Blade 6048) with QDR

- 42U, 24" wide integrated rack
- Four 12-blade shelves (24 Nodes / Shelf)
- Each shelf has a NEM with:
  - 2 36-Port IS4 Switches
  - 8 12x IB cables
- Front-to-rear airflow
- 91% Efficient power supplies
- Redundant power & cooling



Sun Magnum M9

High density, high scalability InfiniBand QDR switch

- Switch Performance
  - 648 ports QDR/DDR/SDR InfiniBand
  - Bisection Bandwidth of 6,480 Tbps
  - 3 Stage internal full Clos network
  - 300ns latency (QDR)
- Line and Fabric Cards
  - 9 Line Cards with connectors
  - 9 Fabric Cards with no connectors
- 11 RU Chassis
  - Mount up to 3 switches in a 19" rack
  - Host based Sun Subnet Manager

"Magnum 9"

Densest 2-tier CLOS switch using 36-p chips

- 576 QDR with 72-p CXP Line Card
  - Max 648 QDR ports
- 41 Tbps total capacity
- 300ns latency (QDR)

Line Cards and Fabric Cards

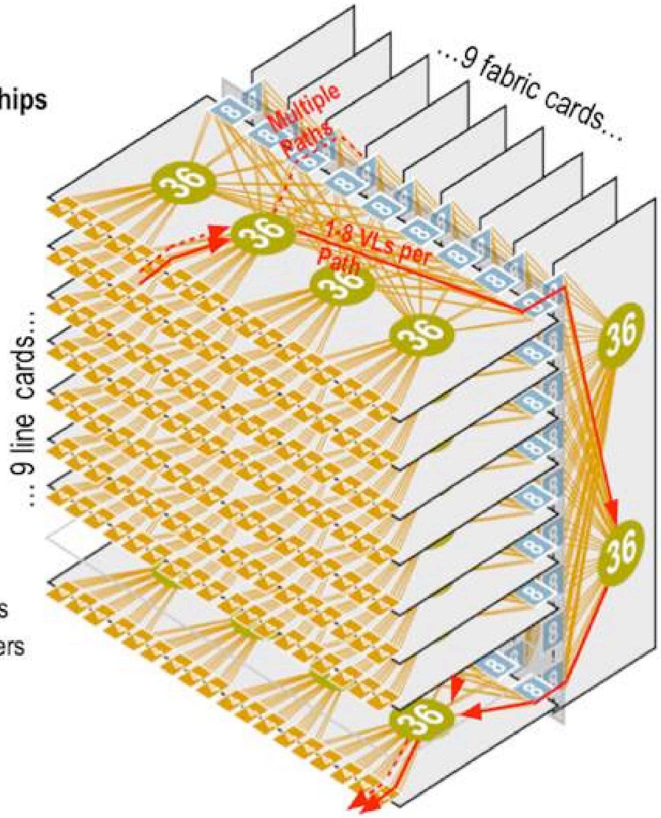
- 8 or 9 Line Cards
- 72-p CXP
- 9 Fabric Cards

11RU 19" Enclosure

- Redundant Power and Cooling
- 7kW power consumption

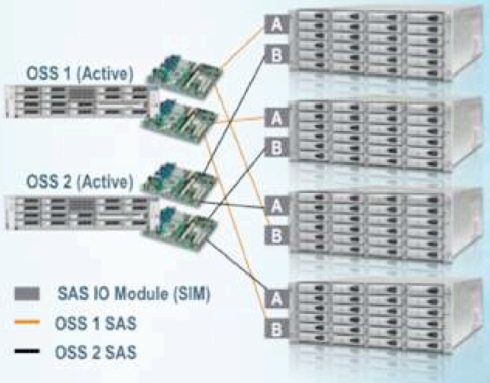
Management

- Redundant hot-swap service processors
- External dual redundant subnet managers



12x Optical Active Cable

HA OSS Module – 9 Pairs (18 OSS)



Primary Paths Connect to SIM A
Secondary Paths Connect to SIM B

Hardware

- 2x Sun Fire X4270s each with
  - Dual, quad-core Intel Nehalem CPUs 2.8GHz, 24GB RAM, dual SAS 2.5" boot drives, Sun 8-port SAS HBA
  - Sun QDR IB-HCA
- 4x Sun Storage J4400 Arrays each with
  - 24x 7200 rpm, 1 TB SATA drives

Software

- Sun Lustre stack & Linux distribution

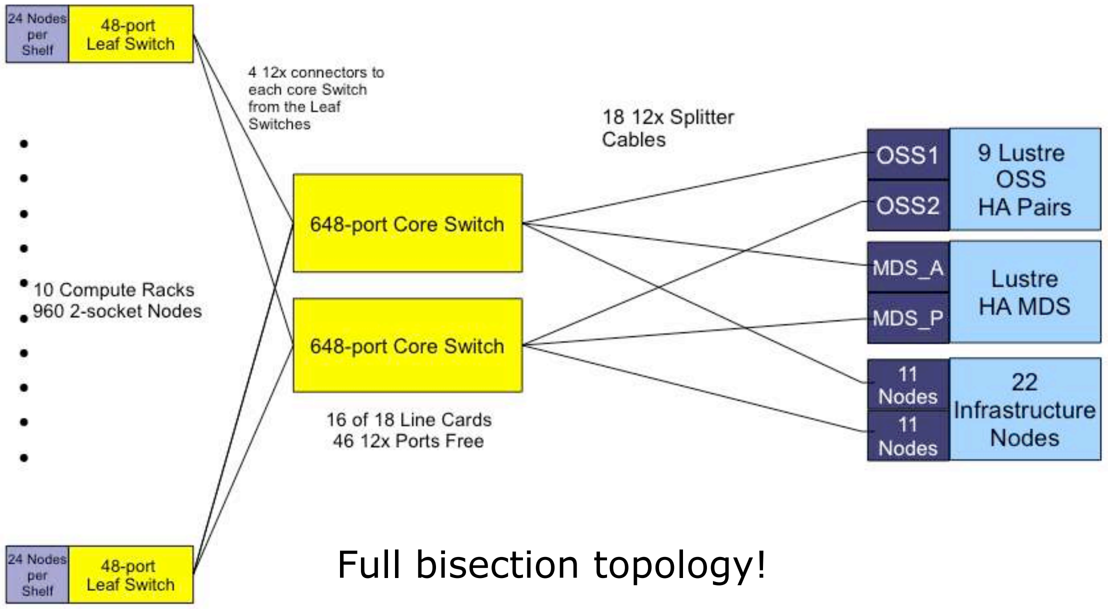
Availability

- Linux RAID 6 for user data
- Lustre OSS configured in active/active pair
- Hot-swap redundant power and cooling

Management

- Integrated LOM Service Processor

Clumeq Overall Architecture - InfiniBand



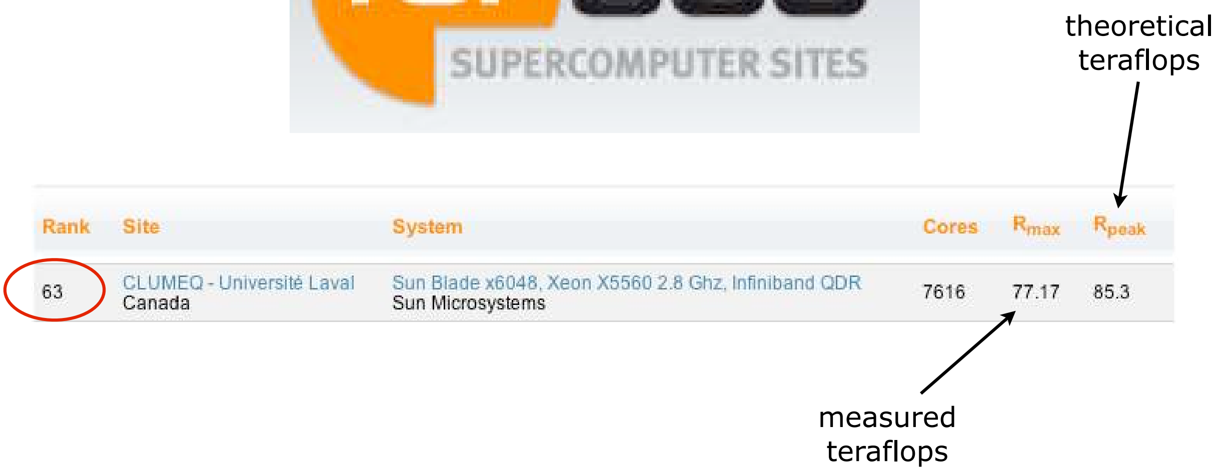
Full bisection topology!

Benchmarking

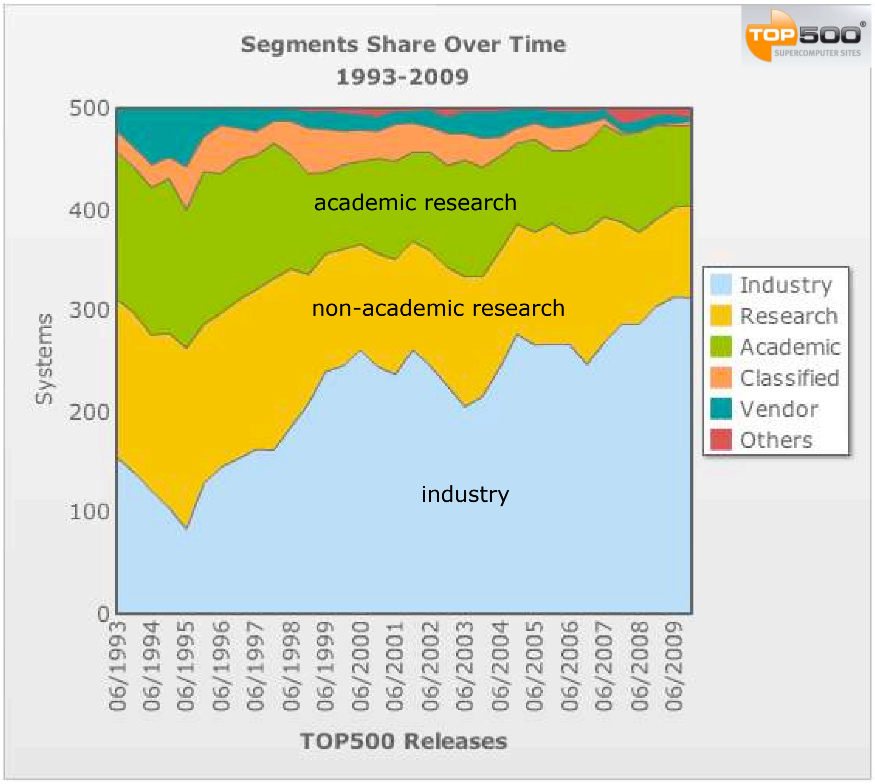
- Compute nodes:
  - ✓ 36.6 < STREAM < 37.9 GB/s
  - ✓ SPECint = 233
  - ✓ 189 < SPECfp < 190
- Interconnect:
  - ✓ MPI ping-pong latency < 2 usec
  - ✓ MPI ping-pong bandwidth > 3.1 GB/s
  - ✓ MPI all-to-all bandwidth > 1.1 GB/s
  - ✓ iPerf > 9.2 Gb/s

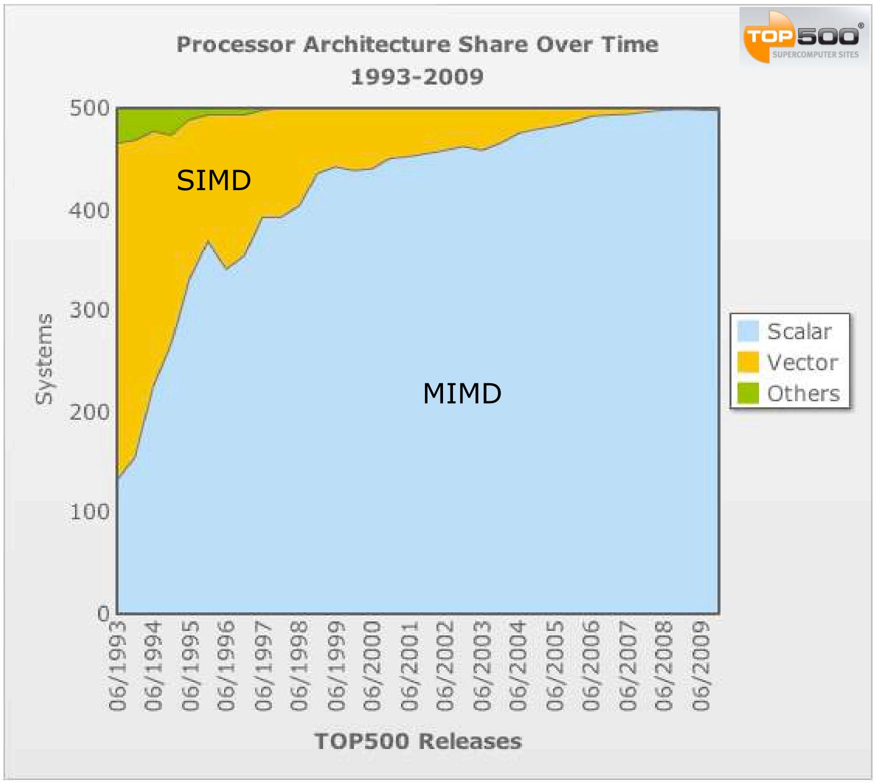
- Lustre file system (18 OSS):
  - ✓ IOR read performance = 33.6 GB/s
  - ✓ IOR write performance = 17.3 GB/s
  - ✓ all over IB
- Boot time: 4 minutes 58 seconds
  - ✓ all over IB
- Max power HPL: 332 kW

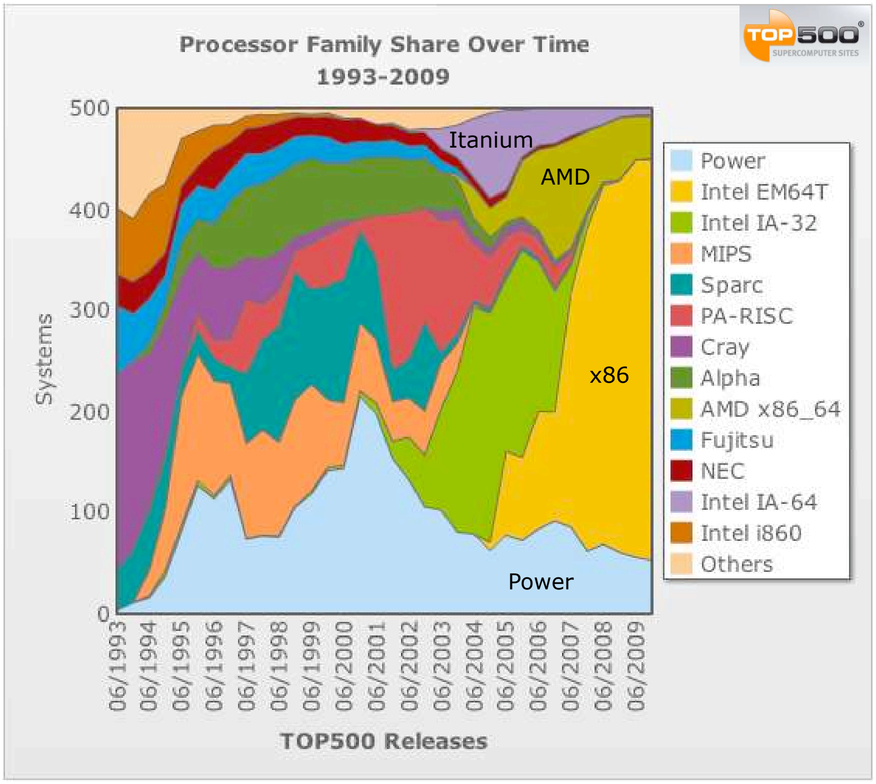
www.top500.org

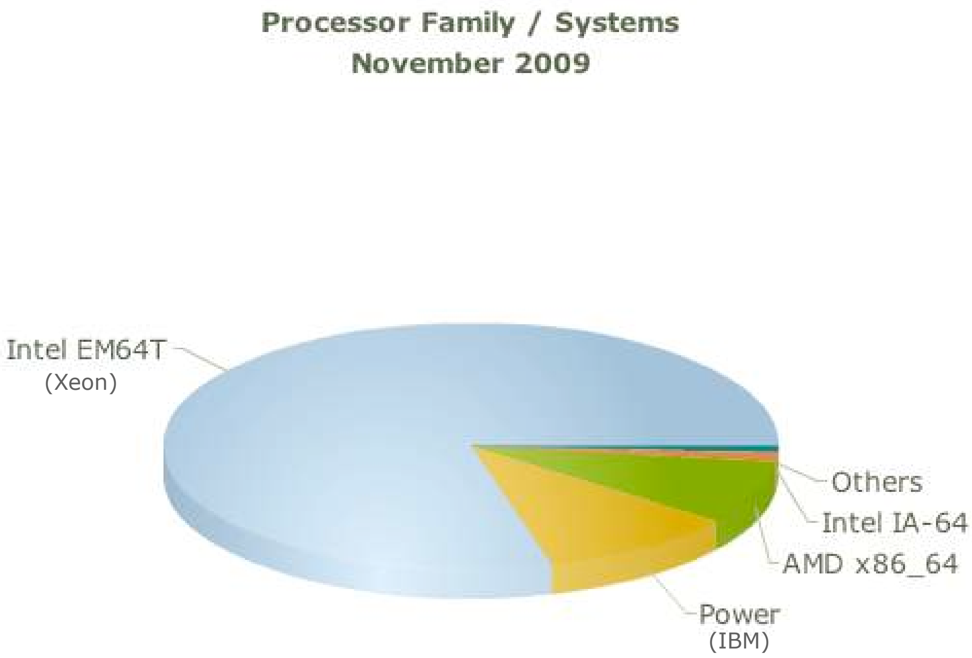


Rank	Site	System	Cores	R_{max}	R_{peak}
63	CLUMEQ - Université Laval Canada	Sun Blade x6048, Xeon X5560 2.8 Ghz, Infiniband QDR Sun Microsystems	7616	77.17	85.3

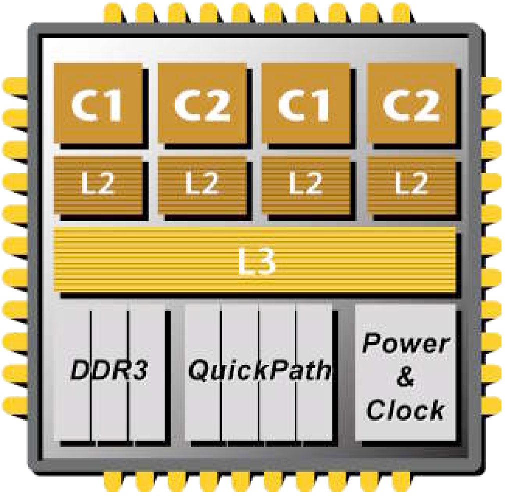

Segments Share Over Time
1993-2009
TOP500 SUPERCOMPUTER SITES
Systems
500
400
300
200
100
0
academic research
non-academic research
industry
06/1993
06/1994
06/1995
06/1996
06/1997
06/1998
06/1999
06/2000
06/2001
06/2002
06/2003
06/2004
06/2005
06/2006
06/2007
06/2008
06/2009
TOP500 Releases
Industry
Research
Academic
Classified
Vendor
Others


Processor Architecture Share Over Time
1993-2009
TOP500 SUPERCOMPUTER SITES
Systems
500
400
300
200
100
0
SIMD
MIMD
06/1993
06/1994
06/1995
06/1996
06/1997
06/1998
06/1999
06/2000
06/2001
06/2002
06/2003
06/2004
06/2005
06/2006
06/2007
06/2008
06/2009
TOP500 Releases
Scalar
Vector
Others

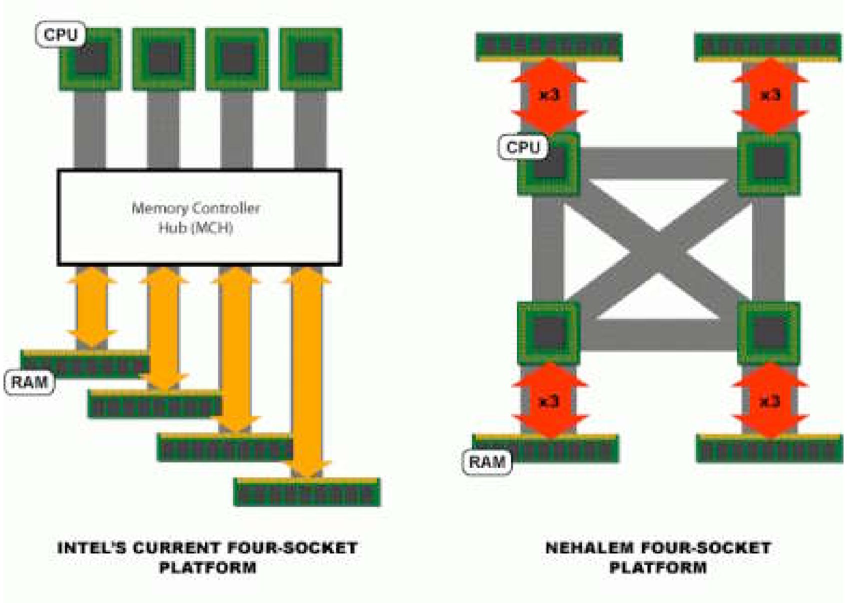

Processor Family Share Over Time
1993-2009
TOP500 SUPERCOMPUTER SITES
500
400
300
200
100
0
Systems
Itanium
AMD
x86
Power
Power
Intel EM64T
Intel IA-32
MIPS
Sparc
PA-RISC
Cray
Alpha
AMD x86_64
Fujitsu
NEC
Intel IA-64
Intel i860
Others
06/1993
06/1994
06/1995
06/1996
06/1997
06/1998
06/1999
06/2000
06/2001
06/2002
06/2003
06/2004
06/2005
06/2006
06/2007
06/2008
06/2009
TOP500 Releases

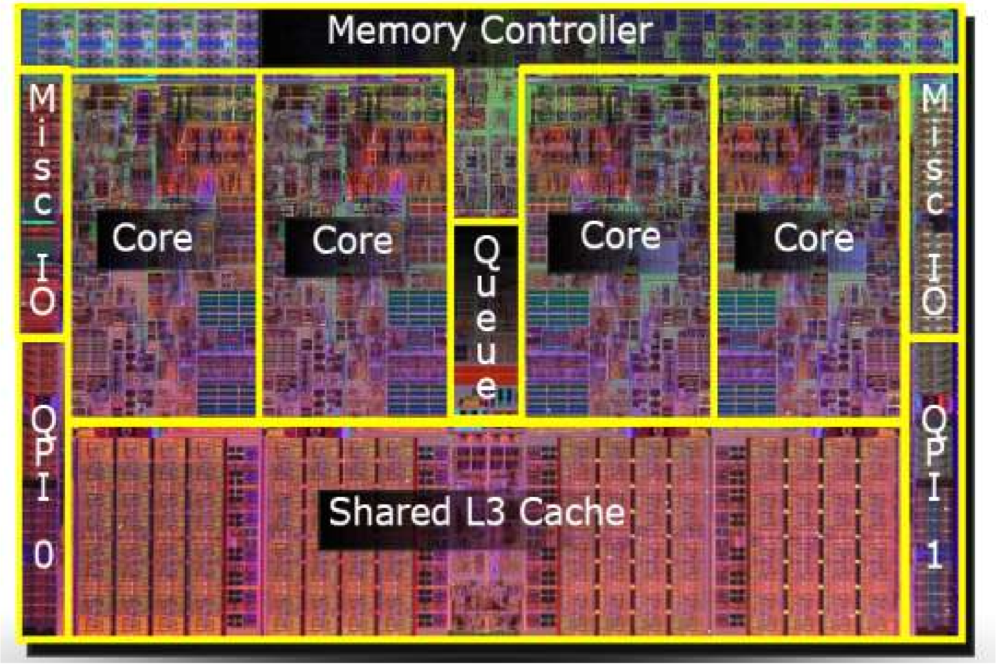

Processor Family / Systems
November 2009
Intel EM64T
(Xeon)
Others
Intel IA-64
AMD x86_64
Power
(IBM)

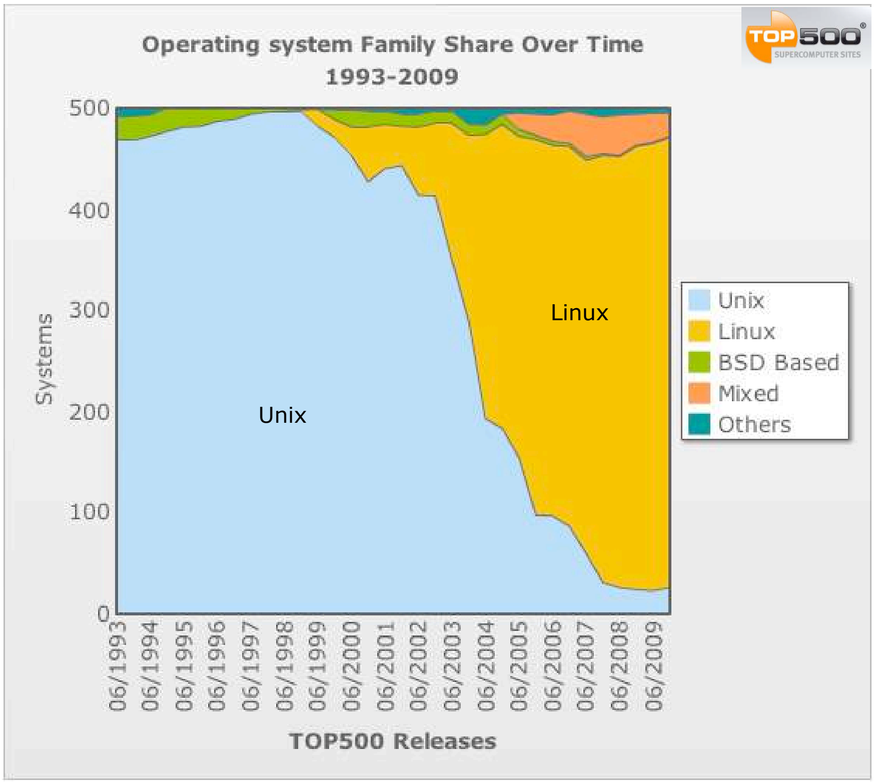
Nehalem architecture

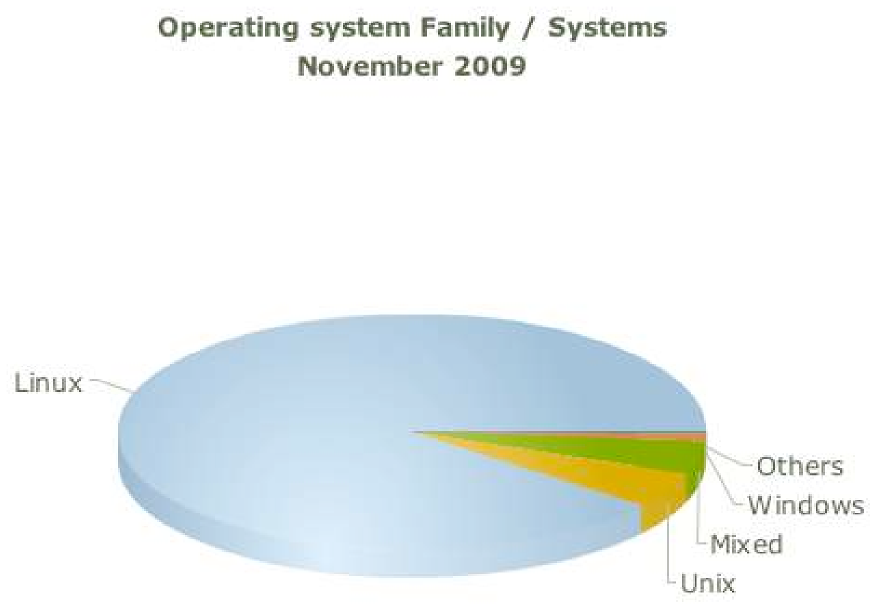


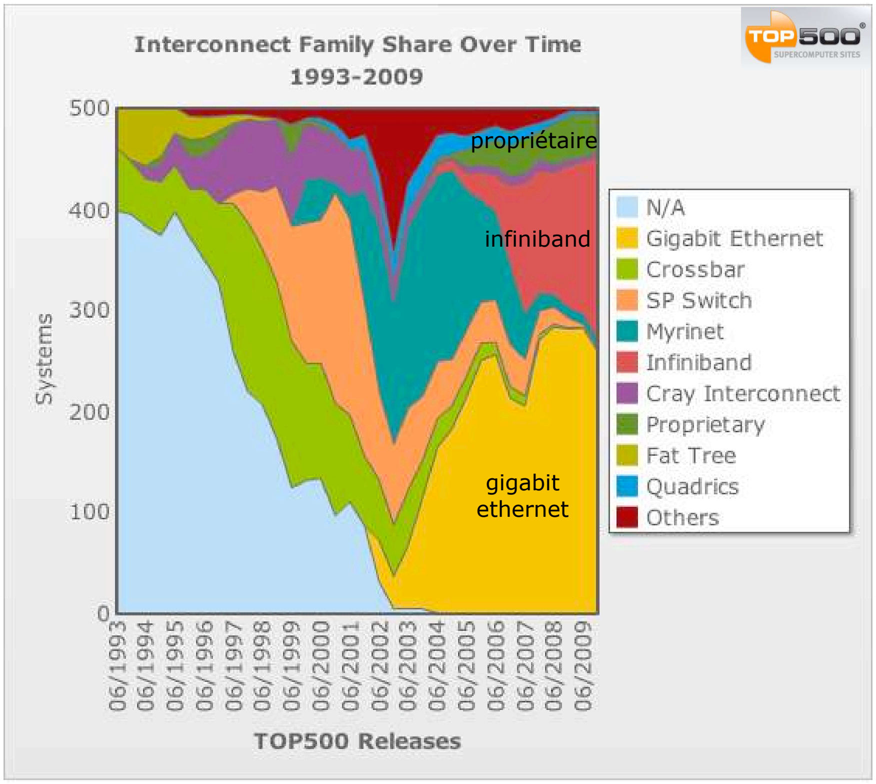
Memory access


CPU
Memory Controller Hub (MCH)
RAM
INTEL'S CURRENT FOUR-SOCKET PLATFORM
x3
x3
CPU
x3
x3
RAM
NEHALEM FOUR-SOCKET PLATFORM


Memory Controller
Misc IO
Core
Core
Queue
Core
Core
Misc IO
QPI 0
Shared L3 Cache
QPI 1


Operating system Family Share Over Time
1993-2009
TOP500 SUPERCOMPUTER SITES
Systems
500
400
300
200
100
0
Unix
Linux
Unix
Linux
BSD Based
Mixed
Others
06/1993
06/1994
06/1995
06/1996
06/1997
06/1998
06/1999
06/2000
06/2001
06/2002
06/2003
06/2004
06/2005
06/2006
06/2007
06/2008
06/2009
TOP500 Releases

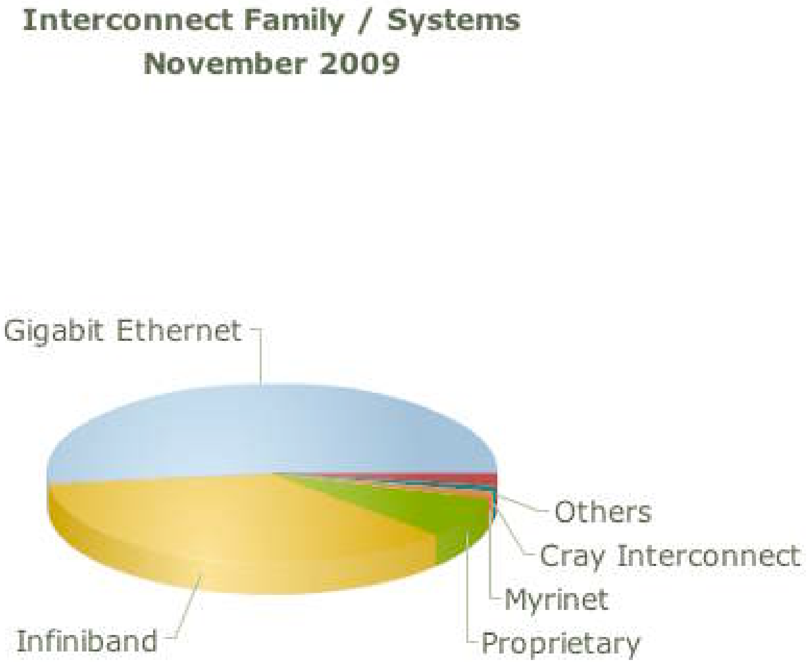




propriétaire

infiniband

gigabit ethernet



Part II: ***tools***

- Hardware: colossus
  - ✓ CLUMEQ
- Software
  - ✓ requirements
  - ✓ survey
- Open BEAGLE

READ tools

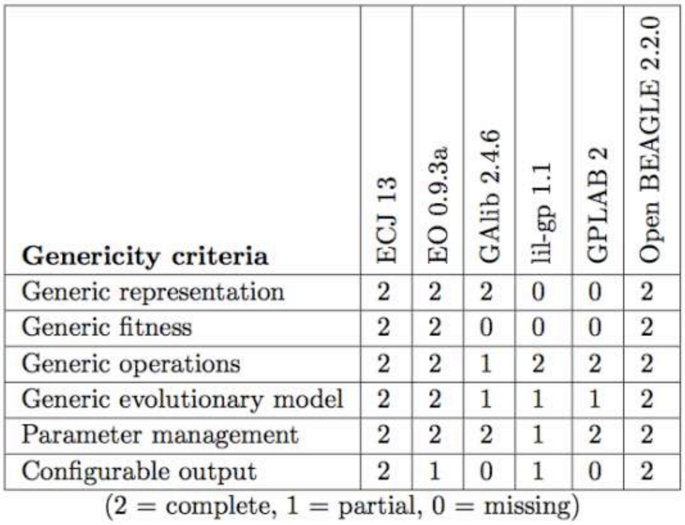
- Research through EA requires quick prototyping
- Tools should be:
  - ✓ simple
  - ✓ flexible
  - ✓ well documented
  - ✓ (reasonably) efficient
- KISS: Keep It Simple and Stupid!

Software requirements

- Code reuse
- Flexibility and adaptability
- Transparency
- Portability
- Ease of use and efficiency

Christian Gagné and Marc Parizeau, "***Genericity in Evolutionary Computation Software Tools: Principles and Case Study***", International Journal on Artificial Intelligence Tools, vol. 15, no 2, pp. 173-194, April 2006.

Survey



Genericity criteria	ECJ 13	EO 0.9.3a	GAlib 2.4.6	lil-gp 1.1	GPLAB 2	Open BEAGLE 2.2.0
Generic representation	2	2	2	0	0	2
Generic fitness	2	2	0	0	0	2
Generic operations	2	2	1	2	2	2
Generic evolutionary model	2	2	1	1	1	2
Parameter management	2	2	2	1	2	2
Configurable output	2	1	0	1	0	2

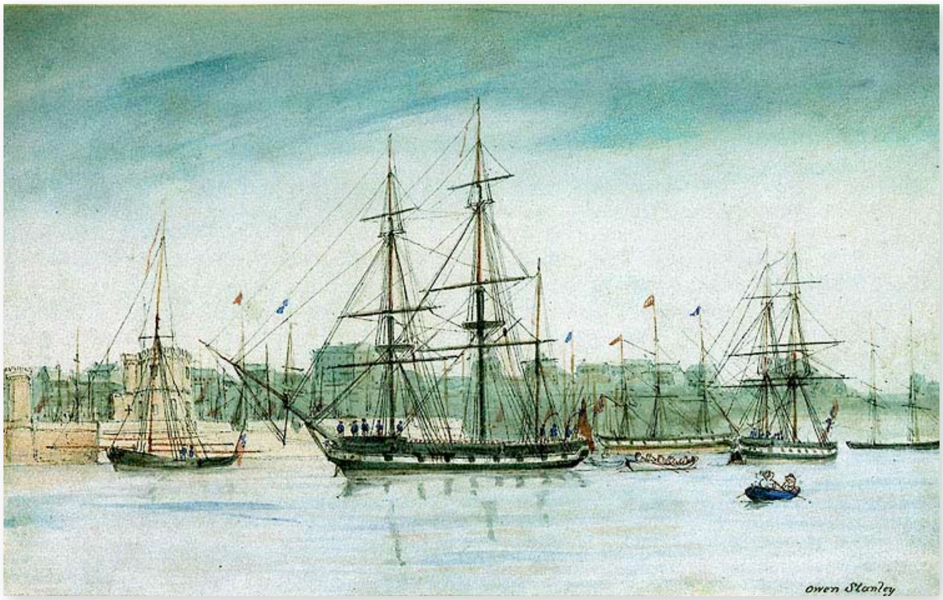
(2 = complete, 1 = partial, 0 = missing)

Open BEAGLE

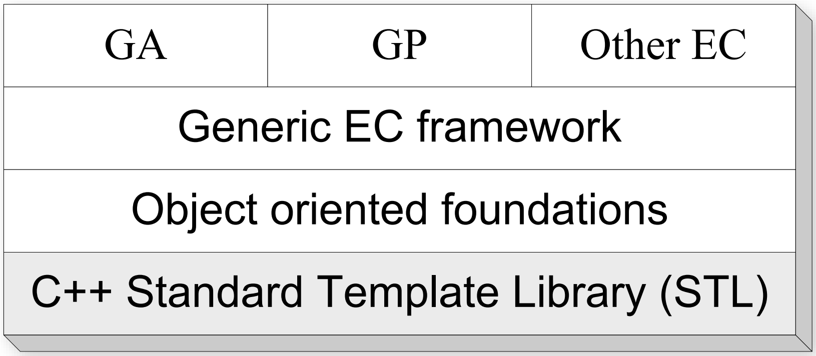
Beagle	**Beagle est un**
Engine is an	**Environnement d'**
Advanced	**Apprentissage**
Genetic	**Génétique**
Learning	**Logiciel**
Environment	**Evolué**

http://beagle.gel.ulaval.ca/

HMS Beagle



Framework



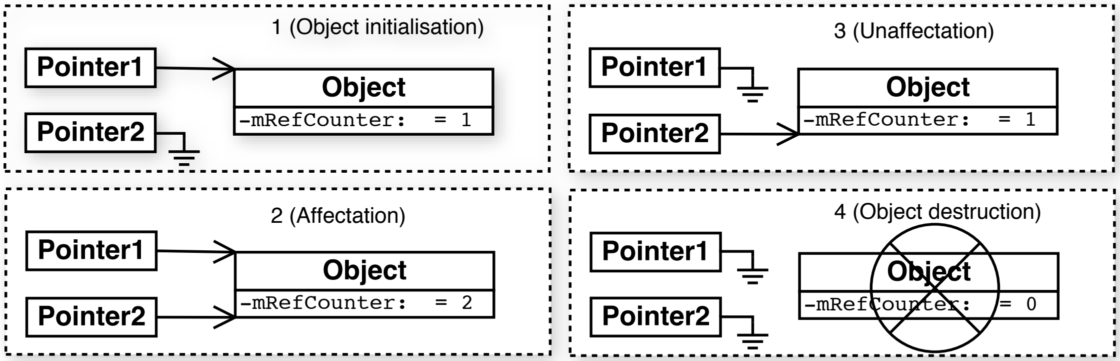
GA | GP | Other EC

Generic EC framework

Object oriented foundations

C++ Standard Template Library (STL)

Intelligent pointer / reference counting



```
template <class T, class BaseType>
class PointerT : public BaseType {
public:
  inline T& operator*();
  inline T* operator->();
};
```

Base class

```
namespace Beagle {
class Object {
public:
  unsigned int getRefCounter() const;
  virtual bool isEqual(const Object&) const;
  virtual bool isLess(const Object&) const;
  virtual void read(XMLNode::Handle&);
  Object*      refer();
  void         unrefer();
  virtual void write(XMLStreamer&) const;
private:
  unsigned int mRefCounter;
};
}
```

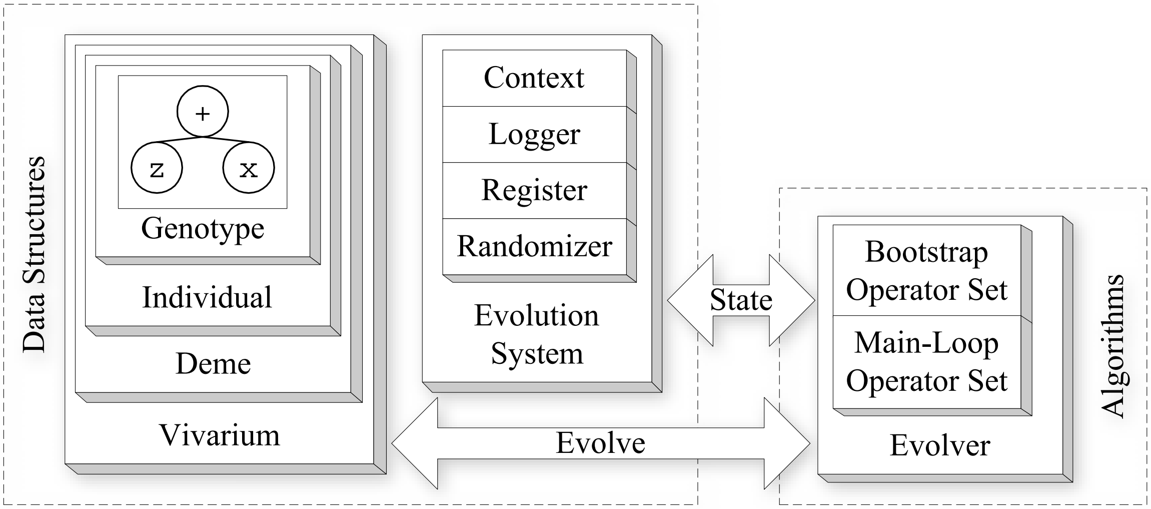
Object factories

```
class Allocator : public Object {
public:
  virtual Object* allocate() const =0;
  virtual Object* clone(const Object&) const =0;
  virtual void    copy(Object&, const Object&) const =0;
};
```

Base type wrappers

C++ name	Wrapper name
bool	Bool
char	Char
double	Double
float	Float
int	Int
long	Long
short	Short
std::string	String
unsigned char	UChar
unsigned int	UInt
unsigned long	ULong
unsigned short	UShort

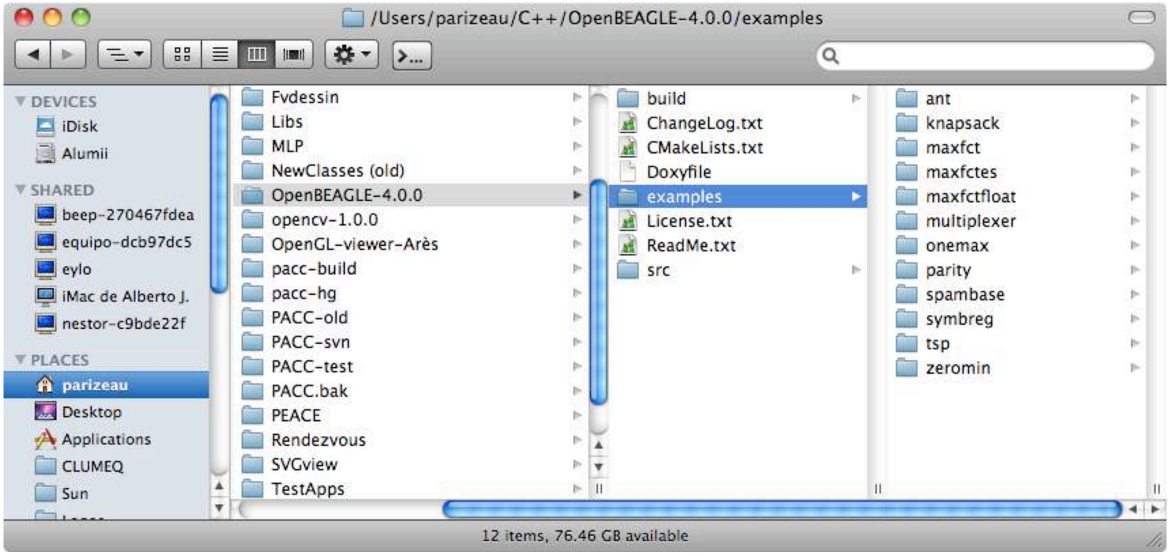
Architecture



Examples

- One max problem
  - ✓ simple bit string representation
  - ✓ find the individual that has the maximum number of "ones"
  - ✓ classical example of *genetic algorithm*
- Symbolic regression
  - ✓ parse tree representation
  - ✓ given a set of points corresponding to an unknown function, find the symbolic expression of this function
  - ✓ classical example of *genetic programming*

BEAGLE examples



/Users/parizeau/C++/OpenBEAGLE-4.0.0/examples

DEVICES
iDisk
Alumii

SHARED
beep-270467fdea
equipo-dcb97dc5
eylo
iMac de Alberto J.
nestor-c9bde22f

PLACES
parizeau
Desktop
Applications
CLUMEQ
Sun

Fvdessin
Libs
MLP
NewClasses (old)
OpenBEAGLE-4.0.0
opencv-1.0.0
OpenGL-viewer-Arès
pacc-build
pacc-hg
PACC-old
PACC-svn
PACC-test
PACC.bak
PEACE
Rendezvous
SVGview
TestApps

build
ChangeLog.txt
CMakeLists.txt
Doxyfile
examples
License.txt
ReadMe.txt
src

ant
knapsack
maxfct
maxfctes
maxfctfloat
multiplexer
onemax
parity
spambase
symbreg
tsp
zeromin

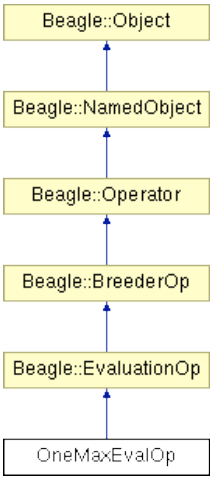
12 items, 76.46 GB available

Example 1
One max problem

- Representation:
  - ✓ bit string
- Objective function:
  - ✓ maximize number of one bits
- Headers:

```
#include "beagle/GA.hpp"
#include "OneMaxEvalOp.hpp"

using namespace std;
using namespace Beagle;
```


Beagle::Object
Beagle::NamedObject
Beagle::Operator
Beagle::BreederOp
Beagle::EvaluationOp
OneMaxEvalOp

```
int main(int argc, char** argv)
{
!  try {
!  !   // 1- Build the system
!  !   System::Handle lSystem = new System;
!  !   // 2- Install the GA bit string package
!  !   const unsigned int lNumberOfBits = 50;
!  !   lSystem->addPackage(new GA::PackageBitString(lNumberOfBits));
!  !   // 3- Add evaluation operator allocator
!  !   lSystem->setEvaluationOp("OneMaxEvalOp", new OneMaxEvalOp::Alloc);
!  !   // 4- Initialize the evolver
!  !   Evolver::Handle lEvolver = new Evolver;
!  !   lEvolver->initialize(lSystem, argc, argv);
!  !   // 5- Create population
!  !   Vivarium::Handle lVivarium = new Vivarium;
!  !   // 6- Launch evolution
!  !   lEvolver->evolve(lVivarium, lSystem);
!  } catch(Exception& inException) {
!  !   inException.terminate(cerr);
!  }
!  return 0;
}
```

```
class OneMaxEvalOp : public Beagle::EvaluationOp
{
public:
   //! OneMaxEvalOp allocator type.
   typedef Beagle::AllocatorT<OneMaxEvalOp,Beagle::EvaluationOp::Alloc> Alloc;
   //! OneMaxEvalOp handle type.
   typedef Beagle::PointerT<OneMaxEvalOp,Beagle::EvaluationOp::Handle> Handle;
   //! OneMaxEvalOp bag type.
   typedef Beagle::ContainerT<OneMaxEvalOp,Beagle::EvaluationOp::Bag> Bag;

   explicit OneMaxEvalOp() : EvaluationOp("OneMaxEvalOp") { }

   virtual Fitness::Handle evaluate(Individual& inIndividual,
                                    Context& ioContext)
   {
      Beagle_AssertM(inIndividual.size() == 1);
      GA::BitString::Handle lBitString = castHandleT<GA::BitString>
                                         (inIndividual[0]);
      unsigned int lCount = 0;
      for(unsigned int i=0; i<lBitString->size(); ++i) {
         if((*lBitString)[i] == true) ++lCount;
      }
      return new FitnessSimple(float(lCount));
   }
};
```

```
void GA::PackageBitString::configure(System& ioSystem)
{
!  Beagle_StackTraceBeginM();
!  Factory& lFactory = ioSystem.getFactory();

!  // Add available operators to the factory
!  lFactory.insertAllocator("GA::CrossoverOnePointBitStrOp",
                            new GA::CrossoverOnePointBitStrOp::Alloc);
!  lFactory.insertAllocator("GA::CrossoverTwoPointsBitStrOp",
                            new GA::CrossoverTwoPointsBitStrOp::Alloc);
!  lFactory.insertAllocator("GA::CrossoverUniformBitStrOp",
                            new GA::CrossoverUniformBitStrOp::Alloc);
!  lFactory.insertAllocator("GA::InitBitStrOp",
                            new GA::InitBitStrOp::Alloc);
!  lFactory.insertAllocator("GA::InitBitStrRampedOp",
                            new GA::InitBitStrRampedOp::Alloc);
!  lFactory.insertAllocator("GA::MutationFlipBitStrOp",
                            new GA::MutationFlipBitStrOp::Alloc);

!  // Set some concept-type associations
!  lFactory.setConcept("CrossoverOp", "GA::CrossoverUniformBitStrOp");
!  lFactory.setConcept("Genotype", "GA::BitString");
!  lFactory.setConcept("InitializationOp", "GA::InitBitStrOp");
!  lFactory.setConcept("MutationOp", "GA::MutationFlipBitStrOp");

!  Beagle_StackTraceEndM("void GA::PackageBitString::configure(System&)");
}
```

Partial observations

- OB is very flexible and very modular
  - ✓ simple to use for predefined EAs
  - ✓ all components can be redefined
  - ✓ many other features not illustrated like introspection, config files, checkpointing, logging, statistics, etc.
- But syntax is sometimes heavy
- Complexity stems from the limitations of the underlying language: C++

Example 2
Symbolic regression

- Representation:
  - ✓ parsed tree of primitives
- Objective:
  - ✓ minimize mean square error between the problem's sample points and the "discovered" function
- Headers:

```
#include "beagle/GP.hpp"
#include "SymbRegEvalOp.hpp"

using namespace std;
using namespace Beagle;
```

Specify the available set of primitives

```
int main(int argc, char *argv[])
{
!  try {
!  !   // 0- Build set of primitives
!  !   GP::PrimitiveSet::Handle lSet = new GP::PrimitiveSet;
!  !   lSet->insert(new GP::Add);
!  !   lSet->insert(new GP::Subtract);
!  !   lSet->insert(new GP::Multiply);
!  !   lSet->insert(new GP::Divide);
!  !   lSet->insert(new GP::Sin);
!  !   lSet->insert(new GP::Cos);
!  !   lSet->insert(new GP::Exp);
!  !   lSet->insert(new GP::Log);
!  !   lSet->insert(new GP::TokenT<Double>("X"));
!  !   lSet->insert(new GP::EphemeralDouble);
       ...
! !  ...
     // 1- Build a system with the "constrained" GP package
! !  System::Handle lSystem = new System;
! !  lSystem->addPackage(new GP::PackageBase(lSet));
! !  lSystem->addPackage(new GP::PackageConstrained);
! !  // 2- Add data set for regression component
! !  lSystem->addComponent(new DataSetRegression);
! !  // 3- Add evaluation operator allocator
! !  lSystem->setEvaluationOp("SymbRegEvalOp",
                              new SymbRegEvalOp::Alloc);
! !  // 4- Initialize the evolver
! !  Evolver::Handle lEvolver = new Evolver;
! !  lEvolver->initialize(lSystem, argc, argv);
! !  // 5- Create population
! !  Vivarium::Handle lVivarium = new Vivarium;
! !  // 6- Launch evolution
! !  lEvolver->evolve(lVivarium, lSystem);
! } catch(Exception& inException) {...}
! return 0;
}
```

```
Fitness::Handle
SymbRegEvalOp::evaluate(GP::Individual& inIndividual,
                        GP::Context& ioContext)
{
! double lSquareError = 0.;
! for(unsigned int i=0; i<mDataSet->size(); i++) {
! !  Beagle_AssertM((*mDataSet)[i].second.size() == 1);
! !  const Double lX((*mDataSet)[i].second[0]);
! !  setValue("X", lX, ioContext);
! !  const Double lY((*mDataSet)[i].first);
! !  Double lResult;
! !  inIndividual.run(lResult, ioContext);
! !  const double lError = lY-lResult;
! !  lSquareError += (lError*lError);
! }
! const double lMSE  = lSquareError / mDataSet->size();
! const double lRMSE = sqrt(lMSE);
! const double lFitness = 1. / (1. + lRMSE);
! return new FitnessSimple(lFitness);
}
```

What about distributed BEAGLE?

- Essentially, you need only to change the package to include new operators
- These operators will split the population into groups of individuals and distribute them to worker nodes in order to evaluate their fitness
- The distribution process use MPI to communicate with worker nodes
- Distribution is thus transparent, but not very flexible

Conclusion

- EAs are fundamentally simple, but writing EA programs is not always easy
  - ✓ frameworks are complex; documentation is not good enough
- Parts of EAs may be compute intensive, but most of the code is complex glue
- Object oriented programming is good, but strongly typed languages are a pain!
  - ✓ higher level languages can significantly increase programmer efficiency and thus lower prototype development time
- Task parallelism must be built-in the framework from the start, not an afterthought!

Part III: *architecture*

- Why Python?
- **DTM**: Distributed task Manager
- **EAP**: Evolutionary Algorithms in Python
- **DTM+EAP=DEAP**: *Distributed Evolutionary Algorithms in Python*
- DEAP optimization and problem solving?

Python language

- Object oriented; fully dynamic
- Coherent syntax
- High level data structures
- Extensive libraries to do mostly anything
- Easy interface to other programming languages like C, C++ or java
- Supports UTF-8 out-of-the-box
- Very efficient glue language!

Python's advantage?

- The language is so powerful and straightforward that you can code your own evolutionary algorithm explicitly (almost like pseudo-code), and control every detail of it!
- Or you can hide as much detail as you want, like how to assign tasks to CPUs in a parallel computer...

- Less lines of codes means:
  - ✓ better readability
  - ✓ less bugs
  - ✓ better documentation!
- Python brings better matlab than Matlab without having to pay for licences!
  - ✓ SciPy, NumPy & matPlotLib
- Want to write platform independent GUIs with...
  - ✓ Qt? FLTK? OpenGL?
- Want to communicate using...
  - ✓ posix sockets? MPI?
- Want to build databases or web services?

Distributed Task Manager (DTM)

- What we want to achieve:
  - ✓ decide at one point in the code that some task(s) should be executed by another process
  - ✓ not worry about where the tasks will execute
  - ✓ not worry about load balancing of tasks
  - ✓ have the option of exploiting transparently anything from a single processor to thousands of them
  - ✓ for debugging, have the possibility of monitoring what is going on!

What about performance?

- No free lunch!
- Real world EC problems have CPU intensive components, but most of the more complex lines of code are just glue representing a small percentage of the total run time
  - ✓ CPU intensive parts should not be coded in Python
- Python interfaces well with other languages
- Programmer/researcher time is much more precious than computer time

What about task granularity?

- No free lunch!
- We leave it to the user to experiment and decide
- Obviously, it is a question of bandwidth and latency
  - ✓ you want relatively small communication overheads

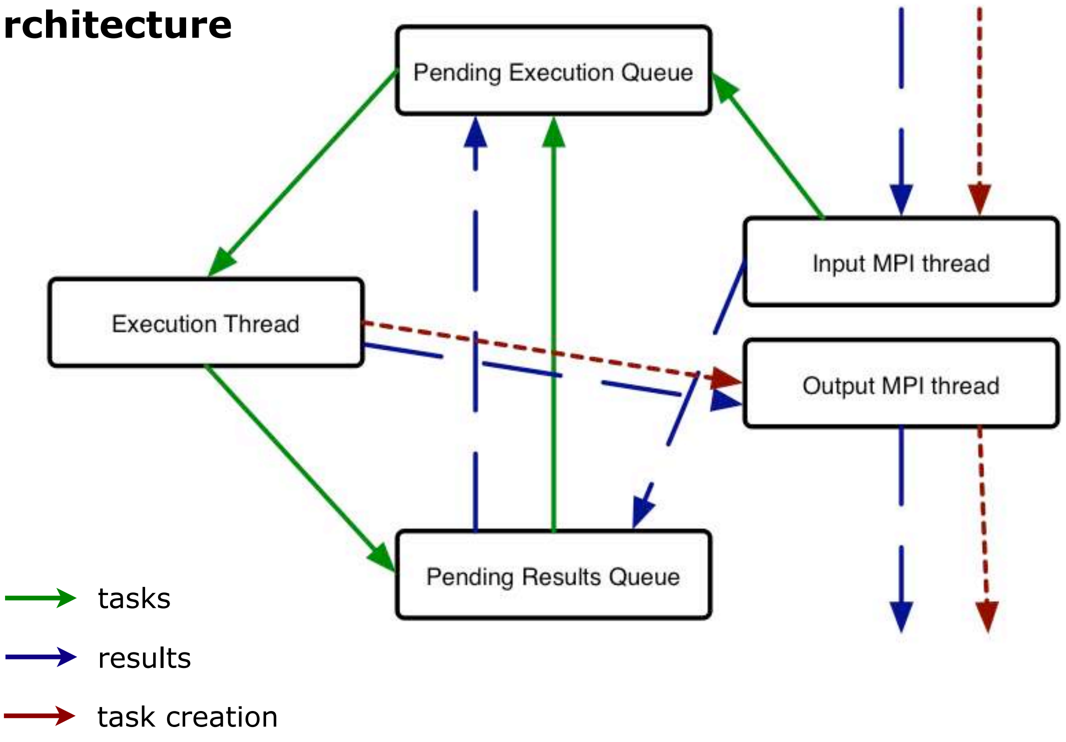
What about existing tools?

- Python has everything that is needed
  - ✓ multithreading classes that run over native OS thread
  - ✓ interface to C/C++ MPI
  - ✓ "pickling" of objects for serialization of everything
  - ✓ just need to write a little bit of glue ;-)
- Many tools have been developed
  - ✓ Intel Cilk++ and Ct
  - ✓ lots of grid stuff
  - ✓ some in Python
- But nothing worth not writing our own

Evil GIL!

- Also called Python's GIL of doom!
  - ✓ GIL=Global Interpreter Lock
  - ✓ threads are pre-empted
  - ✓ but the interpreter cannot run them in parallel on multicore computers
- Solution:
  - ✓ use multiprocessing; one process per core
  - ✓ in a "share nothing" architecture
  - ✓ using message passing

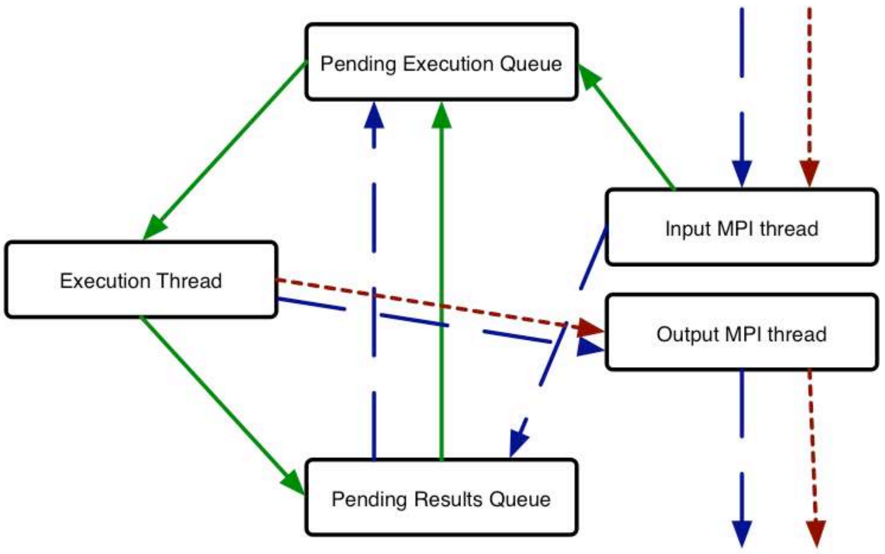
DTM architecture



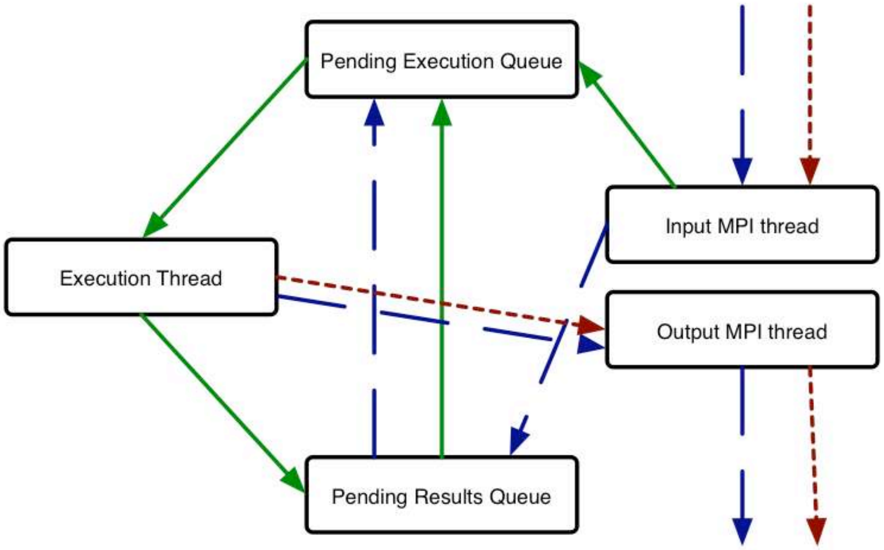
class Task

- Contains...
  - ✓ a unique ID
  - ✓ the ID of its parent
  - ✓ a task type label
  - ✓ a creation, start, and ending time stamp
- Has a run method that receives an argument list

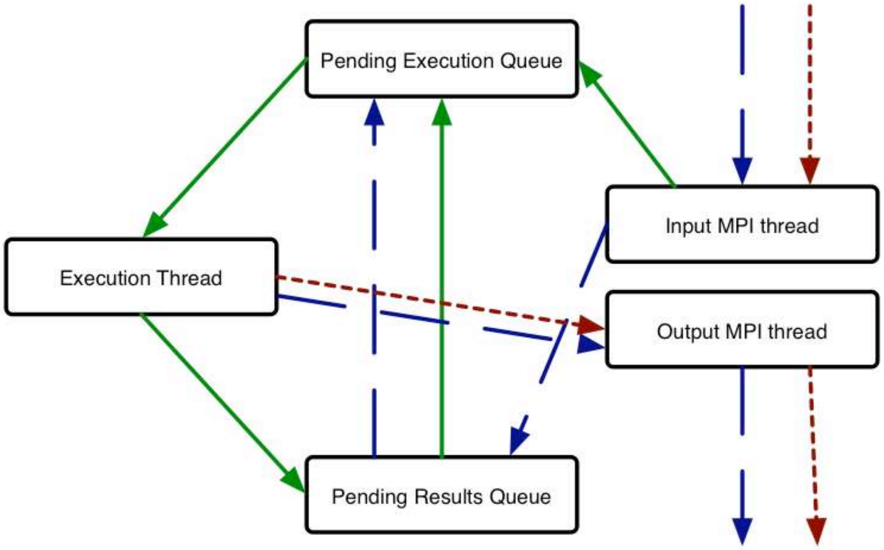
- The "execution thread" handles the currently running task



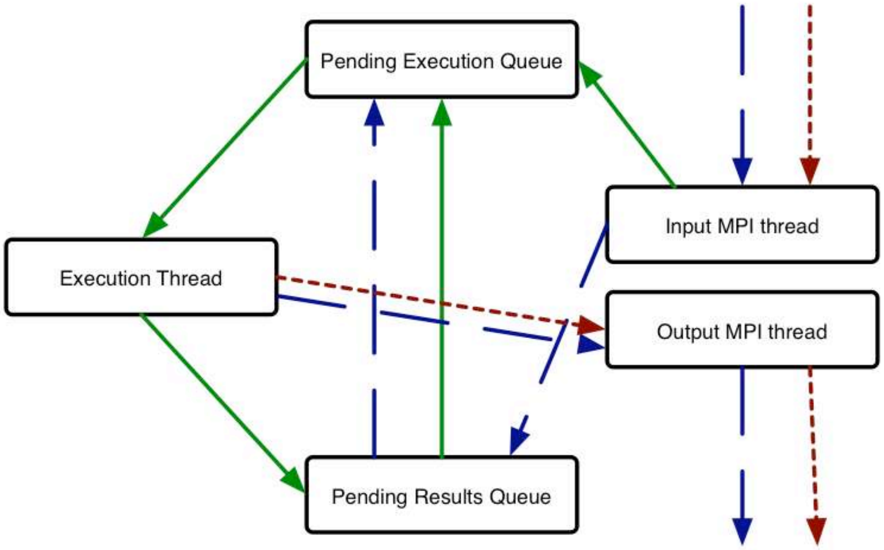
- The "pending execution queue" contains the tasks that are waiting for execution
- It is a priority queue



- The "pending results queue" contains the tasks that have been halted, because they await some result(s)



- The "input/output MPI threads" respectively run the MPI receive/send commands



User interface example

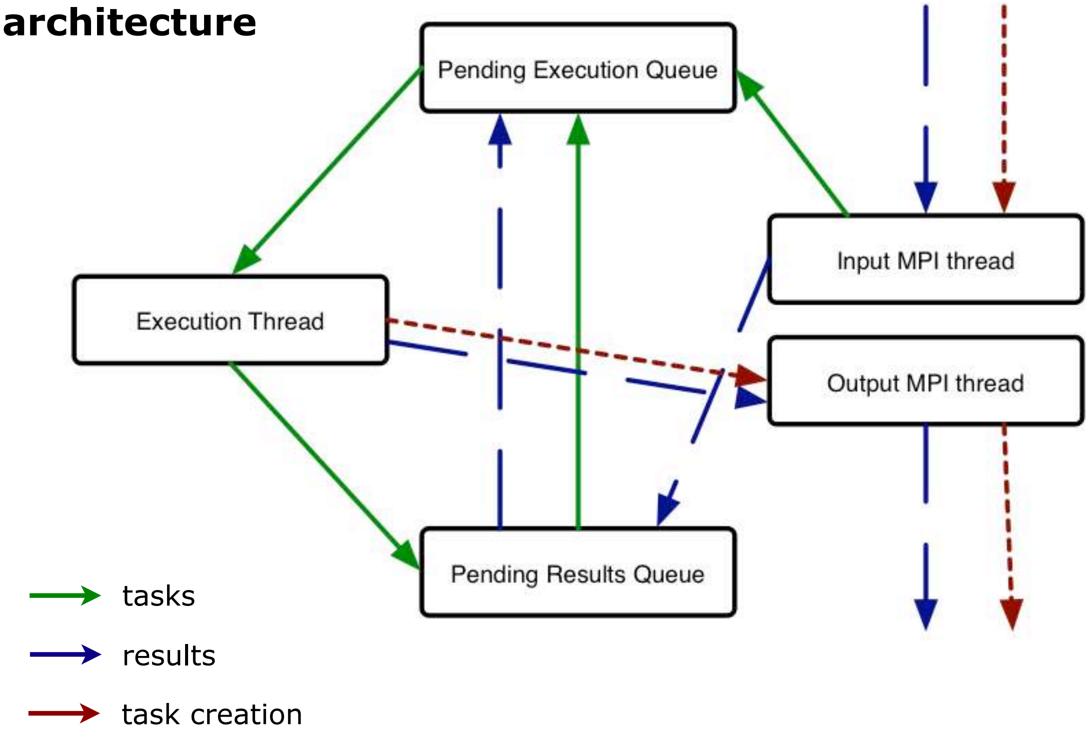
- Initializations
- If MPI rank = 1
  - ✓ do some more initialization
  - ✓ launch root task
  - ✓ for example:

```
dtm.spawn(distributedGA, lTools, lPop, 0.5, 0.2, 40)
```

- Inside the root task (for example):

```
lChilds = [dtm.spawn(toolbox.evaluate, lInd) for lInd
in population]
lData = yield ('waitFor', lChilds)
```

DTM architecture



Load balancing

- Currently, once a task starts executing within a given process, it will remain on that process until completion
- But when a task is spawn, it is randomly assigned to one of the processes that have lower loads
- Each a process communicates with another process, they exchange historical load statistics
- For tasks in the pending execution queue, the load is estimated using the task type label
  - ✓ it is assumed that task with equal labels have similar run times

Evolutionary Algorithms in Python (EAP)

- Fitness
  - ✓ just an array of floats
- Individual
  - ✓ just a sequence (list) of stuff, and a fitness
- Population
  - ✓ just a set (list) of either individuals or sub-populations (demes)
- Toolbox
  - ✓ just a bunch of registered operators that can be used by the evolutionary algorithm

Fitness

- A class derived from a simple array of floats

```
class Fitness(array.array):
  def isValid(self):
  def invalidate(self):
  def isDominated(self, other):
```

  - ✓ works the same way for single or multiple values (objectives)

Maximize or minimize?

- The Fitness constructor has an optional argument to assign weights to the different objectives
  - ✓ `+1` (default) indicates that the corresponding component should be maximized
  - ✓ `-1` indicates minimization

```
def __init__(self, weights=(-1.0,)):
  self.mWeights = array.array('d', weights)
```

Individual

- A container class derived from a list of things; the kind of "things" being specified by a generator function...

```
class Individual(list):
  def __init__(self, size=0, generator=None,
               fitness=None):
     if fitness is not None:
        self.mFitness = fitness()
     for i in xrange(size):
        self.append(generator.next())
```

Population

- A container class derived from a list of "things"; the kind of things being specified by an object...

```
class Population(list):
def __init__(self, size=0, generator=None):
  for i in xrange(size):
    self.append(generator())
```

Toolbox

- Just a factory to manufacture evolutionary methods:

```
class Toolbox(object):
  def register(self, methodName, method, *args, **kargs):
  def unregister(self, methodName):
```

- Toolset examples:

```
def tournSel(individuals, n, tournSize=2):
def wheelSel(individuals, n):
def onePointCx(indOne, indTwo):
def twoPointsCx(indOne, indTwo):
def pmxCx(indOne, indTwo):
def flipBitMut(individual, prob):
def gaussMut(individual, sigma, prob):
```

```
import eap.base as base
import eap.toolbox as toolbox

# create toolbox
lTools = toolbox.Toolbox()

# populate toolbox with fitness, individual,
# and population creators
lTools.register('fitness', base.Fitness,
                weights=(1.0,))
lTools.register('individual', base.Individual,
                size=100, fitness=lTools.fitness,
                generator=base.booleanGenerator())
lTools.register('population', base.Population,
                size=300, generator=lTools.individual)

# create the initial population
lPop = lTools.population()
```

```
# define the evaluation method
def evalOneMax(individual):
  if not individual.mFitness.isValid():
    individual.mFitness.append(individual.count(True))

# populate toolbox with evolutionary operators
lTools.register('evaluate', evalOneMax)

lTools.register('crossover', toolbox.twoPointsCx)

lTools.register('mutate', toolbox.flipBitMut,
                flipIndxPb=0.05)

lTools.register('select', toolbox.tournSel,
                tournSize=3)

# Evaluate the initial population
map(lTools.evaluate, lPop)
```

simple evolution loop

```
CXPB, MUTPB, NGEN = (0.5, 0.2, 40)

for g in range(NGEN):
    print 'Generation', g

    lPop[:] = lTools.select(lPop, n=len(lPop))

    # Apply crossover and mutation
    for i in xrange(1, len(lPop), 2):
        if random.random() < CXPB:
            lPop[i - 1], lPop[i] = lTools.crossover(lPop[i - 1], lPop[i])
    for i in xrange(len(lPop)):
        if random.random() < MUTPB:
            lPop[i] = lTools.mutate(lPop[i])

    # Evaluate the population
    map(lTools.evaluate, lPop)

    # Gather all the fitnesses in one list and print the stats
    lFitnesses = [lInd.mFitness[0] for lInd in lPop]
    print '\tMin Fitness :', min(lFitnesses)
    print '\tMax Fitness :', max(lFitnesses)
    print '\tMean Fitness :', sum(lFitnesses)/len(lFitnesses)

print 'End of evolution'
```

OneMax short example

```
import eap.base as base
import eap.algorithms as algorithms
import eap.toolbox as toolbox

def evalOneMax(individual):
    if not individual.mFitness.isValid():
        individual.mFitness.append(individual.count(True))

lTools = toolbox.Toolbox()
lTools.register('fitness', base.Fitness, weights=(1.0,))
lTools.register('individual', base.Individual, size=100,
                fitness=lTools.fitness, generator=base.booleanGenerator())
lTools.register('population', base.Population, size=300,
                generator=lTools.individual)
lTools.register('evaluate', evalOneMax)
lTools.register('crossover', toolbox.twoPointsCx)
lTools.register('mutate', toolbox.flipBitMut, flipIndxPb=0.05)
lTools.register('select', toolbox.tournSel, tournSize=3)

lPop = lTools.population()
algorithms.simpleGA(lTools, lPop, cxPb=0.5, mutPb=0.2, nGen=40)
```

```
def simpleGA(toolbox, population, cxPb, mutPb, nGen):
   # Evaluate the initial population
   map(toolbox.evaluate, population)
   # run the evolution loop
   for g in range(nGen):
      print 'Generation', g
      population[:] = toolbox.select(population, n=len(population))
      # Apply crossover and mutation
      for i in xrange(1, len(population), 2):
        if random.random() < cxPb:
        population[i - 1], population[i] = toolbox.crossover(population
        [i-1], population[i])
      for i in xrange(len(population)):
        if random.random() < mutPb:
        population[i] = toolbox.mutate(population[i])
      # Evaluate the population
      map(toolbox.evaluate, population)
      # Gather all of the fitness values in one list and print
      statistics
      lFitnesses = [lInd.mFitness[0] for lInd in population]
      print '\tMin Fitness :', min(lFitnesses)
      print '\tMax Fitness :', max(lFitnesses)
      print '\tMean Fitness :', sum(lFitnesses)/len(lFitnesses)
   print 'End of evolution'
```

DTM+EAP = DEAP

```
from mpi4py import MPI
import eap.base as base
import eap.toolbox as toolbox

def evalOneMax(individual):
   if not individual.mFitness.isValid():
      yield individual.count(True)

if MPI.COMM_WORLD.Get_rank() == 0:
   lTools = toolbox.Toolbox()
   lTools.register('fitness', base.Fitness, weights=(1.0,))
   lTools.register('individual', base.Individual, size=100,\
                  fitness=lTools.fitness, generator=base.booleanGenerator())
   lTools.register('population', base.Population, size=300,\
                  generator=lTools.individual)

   lTools.register('evaluate', evalOneMax)
   lTools.register('crossover', toolbox.twoPointsCx)
   lTools.register('mutate', toolbox.flipBitMut, flipIndxPb=0.05)
   lTools.register('select', toolbox.tournSel, tournSize=3)

   lPop = lTools.population()
   dtm.spawn(distributedGA, lTools, lPop, 0.5, 0.2, 40)
```

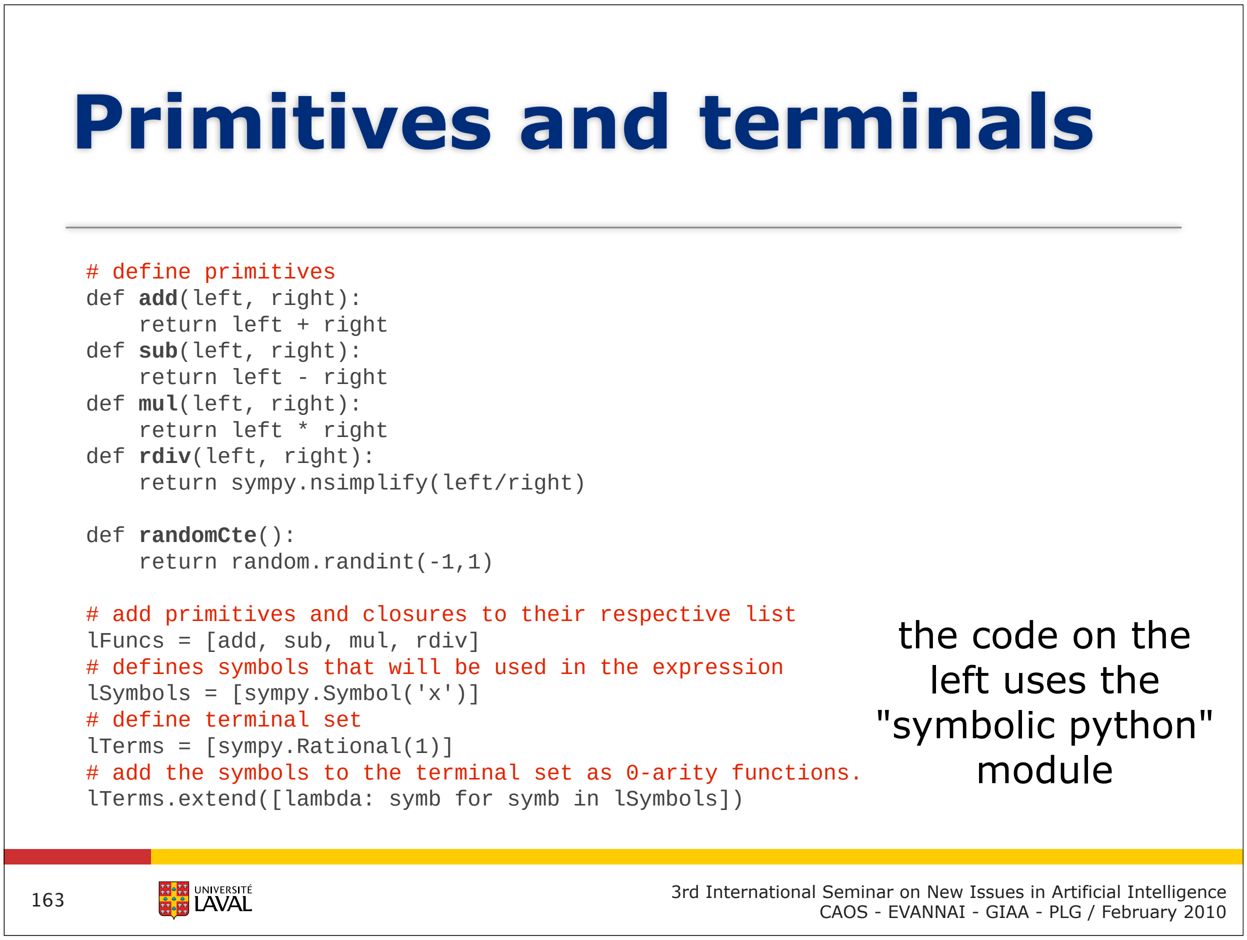
```
def distributedGA(toolbox, population, cxPb, mutPb, nGen):
    # Evaluate the population
    map(toolbox.evaluate, population)
    # Begin the evolution
    for g in range(nGen):
        print 'Generation', g
        population[:] = toolbox.select(population, n=len(population))
        # Apply crossover and mutation
        for i in xrange(1, len(population), 2):
            if random.random() < cxPb:
            population[i - 1], population[i] = toolbox.crossover(population[i - 1],
                                                                  population[i])
        for i, ind in enumerate(population):
            if random.random() < mutPb:
            population[i] = toolbox.mutate(ind)
        # Distribute the evaluation
        lChilds = [dtm.spawn(toolbox.evaluate, lInd) for lInd in population]
        lData = yield ('waitFor', lChilds)
        for i, lID in enumerate(lChilds):
            population[i].mFitness.append(lData[lID])
        # Gather all fitness values in one list and print statistics
        lFitnesses = [lInd.mFitness[0] for lInd in population]
        print '\tMin Fitness :', min(lFitnesses)
        print '\tMax Fitness :', max(lFitnesses)
        print '\tMean Fitness :', sum(lFitnesses)/len(lFitnesses)

    print 'End of evolution'
```

What about GP?

- Need to...
  - ✓ build the set of primitives
  - ✓ build the set of terminals
  - ✓ define the evaluation function
  - ✓ register everything
  - ✓ and call the "simpleGA" algorithm
- Import modules:

```
import sympy
import random
import math
import eap.base as base
import eap.toolbox as toolbox
import eap.algorithms as algorithms
```



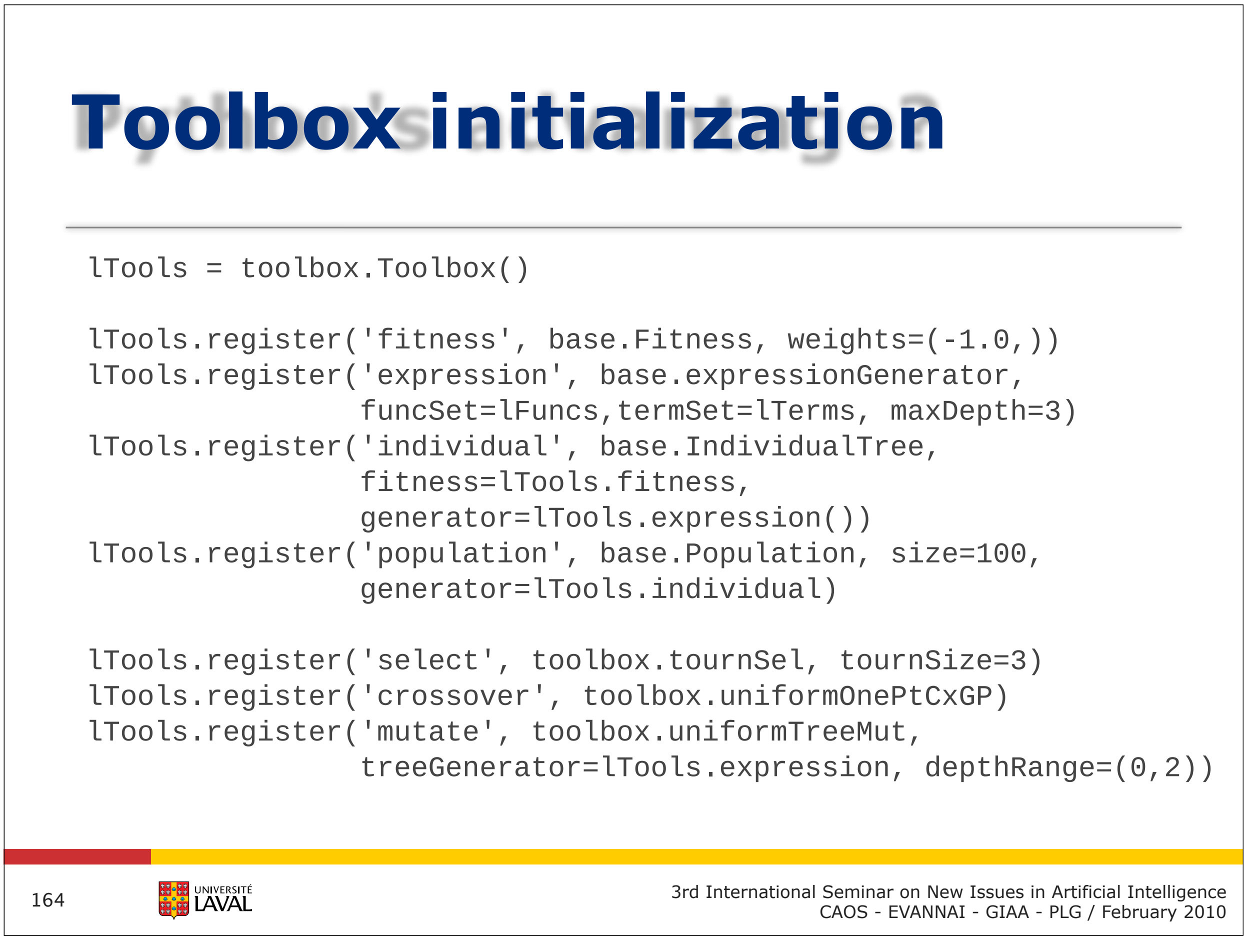
Primitives and terminals

```
# define primitives
def add(left, right):
    return left + right
def sub(left, right):
    return left - right
def mul(left, right):
    return left * right
def rdiv(left, right):
    return sympy.nsimplify(left/right)

def randomCte():
    return random.randint(-1,1)

# add primitives and closures to their respective list
lFuncs = [add, sub, mul, rdiv]
# defines symbols that will be used in the expression
lSymbols = [sympy.Symbol('x')]
# define terminal set
lTerms = [sympy.Rational(1)]
# add the symbols to the terminal set as 0-arity functions.
lTerms.extend([lambda: symb for symb in lSymbols])
```

the code on the left uses the "symbolic python" module



Toolbox initialization

```
lTools = toolbox.Toolbox()

lTools.register('fitness', base.Fitness, weights=(-1.0,))
lTools.register('expression', base.expressionGenerator,
                funcSet=lFuncs,termSet=lTerms, maxDepth=3)
lTools.register('individual', base.IndividualTree,
                fitness=lTools.fitness,
                generator=lTools.expression())
lTools.register('population', base.Population, size=100,
                generator=lTools.individual)

lTools.register('select', toolbox.tournSel, tournSize=3)
lTools.register('crossover', toolbox.uniformOnePtCxGP)
lTools.register('mutate', toolbox.uniformTreeMut,
                treeGenerator=lTools.expression, depthRange=(0,2))
```

```
def evalSymbReg(individual, symbols):
  if not individual.mFitness.isValid():
    # Simplify the expression by collecting the terms
    expr = individual.evaluate()
    # Transform expression in a callable function
    lFuncExpr = sympy.lambdify(symbols, expr)
    lDiff = 0
    # Evaluate the sum of squared difference
    # real function : x**4 + x**3 + x**2 + x + 1
    for x in xrange(-100,100):
       x = x/100.
       try:
          lDiff += (lFuncExpr(x)-(x**4 + x**3 +
                   x**2 + x + 1))**2
       except ZeroDivisionError:
          lDiff += ((x**4 + x**3 + x**2 + x + 1))**2

    individual.mFitness.append(lDiff)
```

DEAP philosophy

- Transparent and minimalist design
  - ✓ not a blackbox design!
  - ✓ not bloated with specialized features,
  - ✓ but generic enough to build sophisticated specialized distributed evolutionary algorithms
  - ✓ you want to visualize your complete evolutionary algorithm on one page
  - ✓ you are exposed to the level of details that you decide
  - ✓ you have complete control if you want it!

To do list

- This is a work in progress...
  - ✓ implement multiobjective and co-evolution
  - ✓ develop other advance algorithms
  - ✓ develop utility functions like checkpointing and logging (easy in Python), etc.
  - ✓ develop monitoring tools for DTM
- Currently working on the project
  - ✓ 1 undergraduate (part-time)
  - ✓ 2 masters (part-time)
- Soon three or four PhDs will be using it for their research projects
- Project started last summer; development is now ramping up quickly!

Questions?