Parco 2009

Lyon, September 1st - 4th, 2009

Skeletons for multi/many core systems

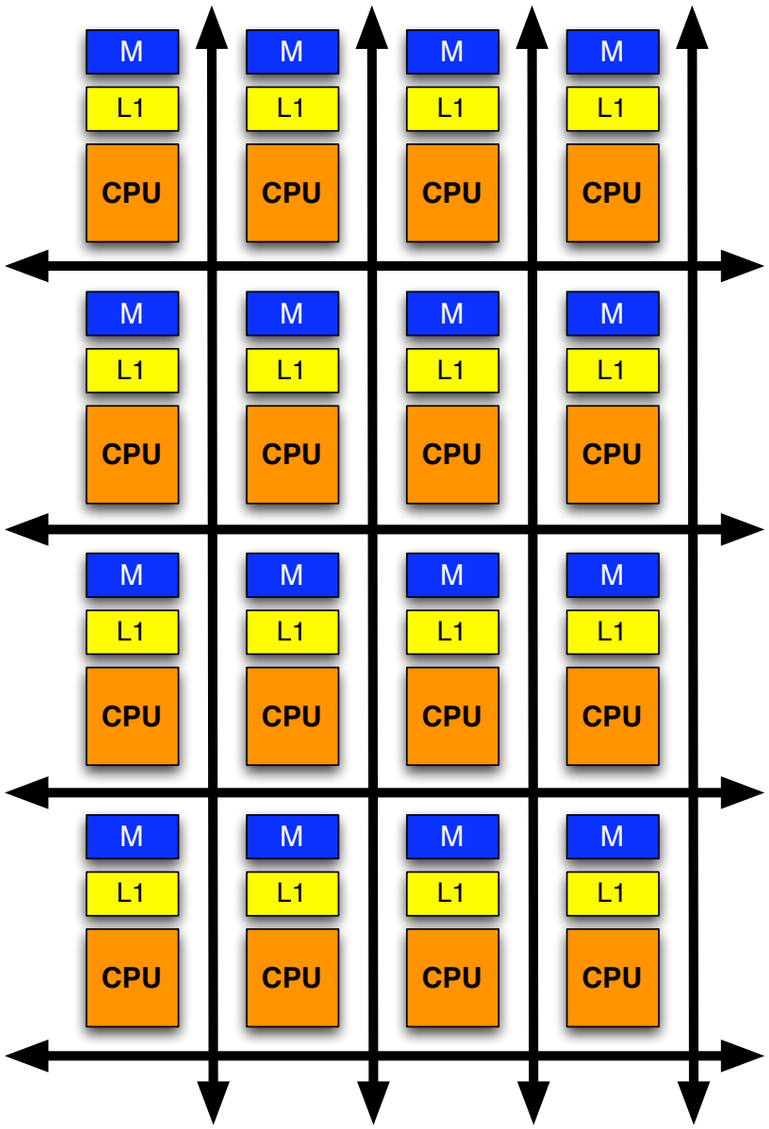
Marco Aldinucci, Univ. of Torino, I

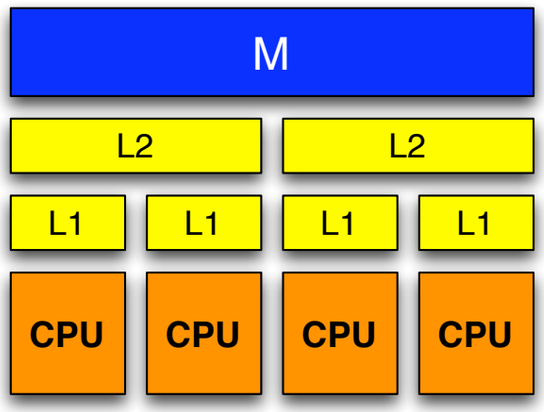
Marco Danelutto, Univ. of Pisa, I

Peter Kilpatrick, Queen's Univ. of Belfast, UK

Motivation

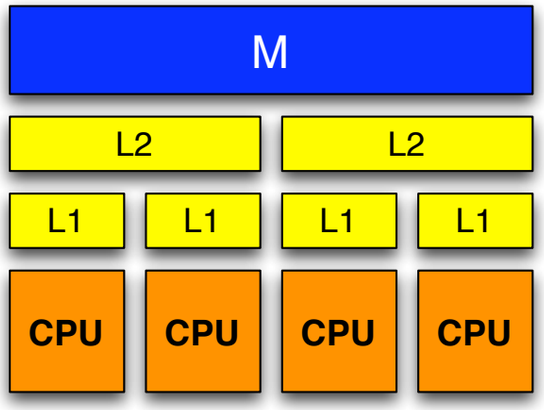
- Skeleton technology recognized successful
  - *Muesli, SkeTo, ASSIST, Muskel, Mallba, GCM-BS*
- Multi-core and many core recent development →
  parallel programs needed

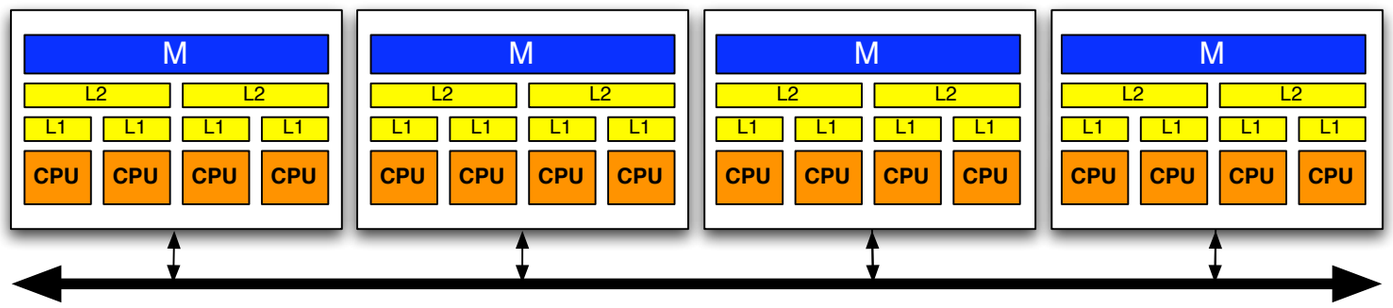




Motivation

- Skeleton technology recognized successful
  - *Muesli, SkeTo, ASSIST, Muskel, Mallba, GCM-BS*
- Multi-core and many core recent development ➙ ***parallel programs needed***



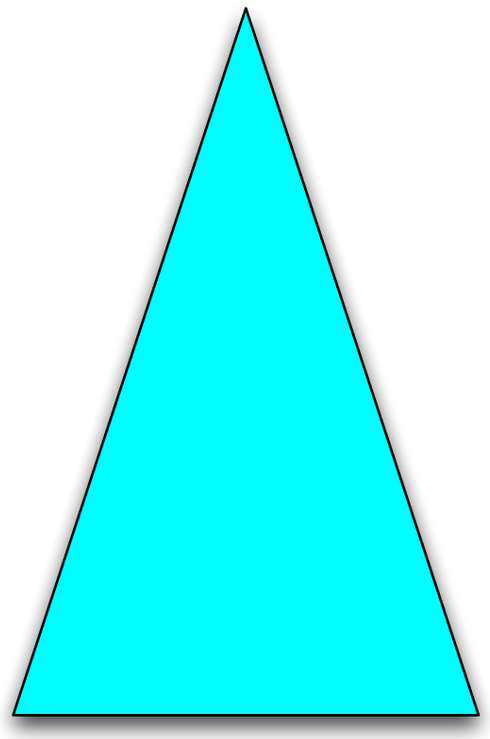


Skeleton concept

- Know, efficient, reusable, parametric parallelism exploitation pattern
- Skeleton framework
- *application programmers use skeletons, system designers implement them (separation of concerns)*
- *nesting (often) allowed (two tier model)*
- *portability across target architectures*
- *performance portability*

Skeleton concept

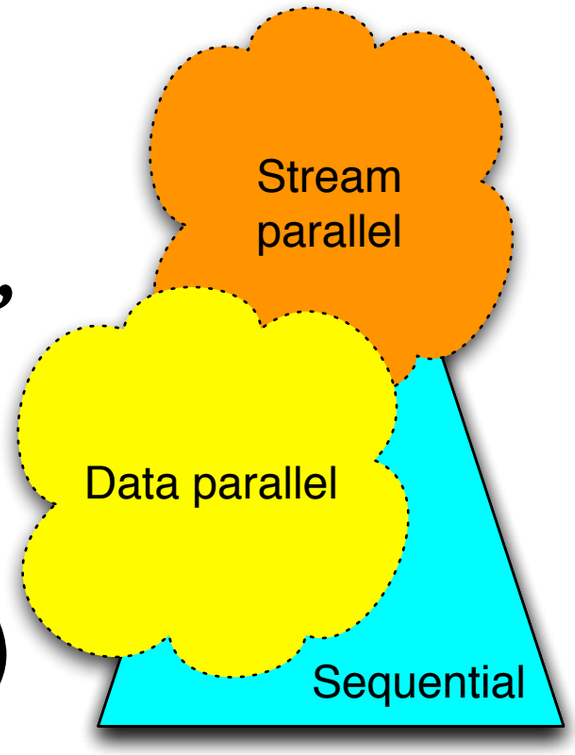
- Know, efficient, reusable, parametric parallelism exploitation pattern
- Skeleton framework
- *application programmers use skeletons, system designers implement them (separation of concerns)*
- *nesting (often) allowed (two tier model)*
- *portability across target architectures*
- *performance portability*



Skeleton tree

Skeleton concept

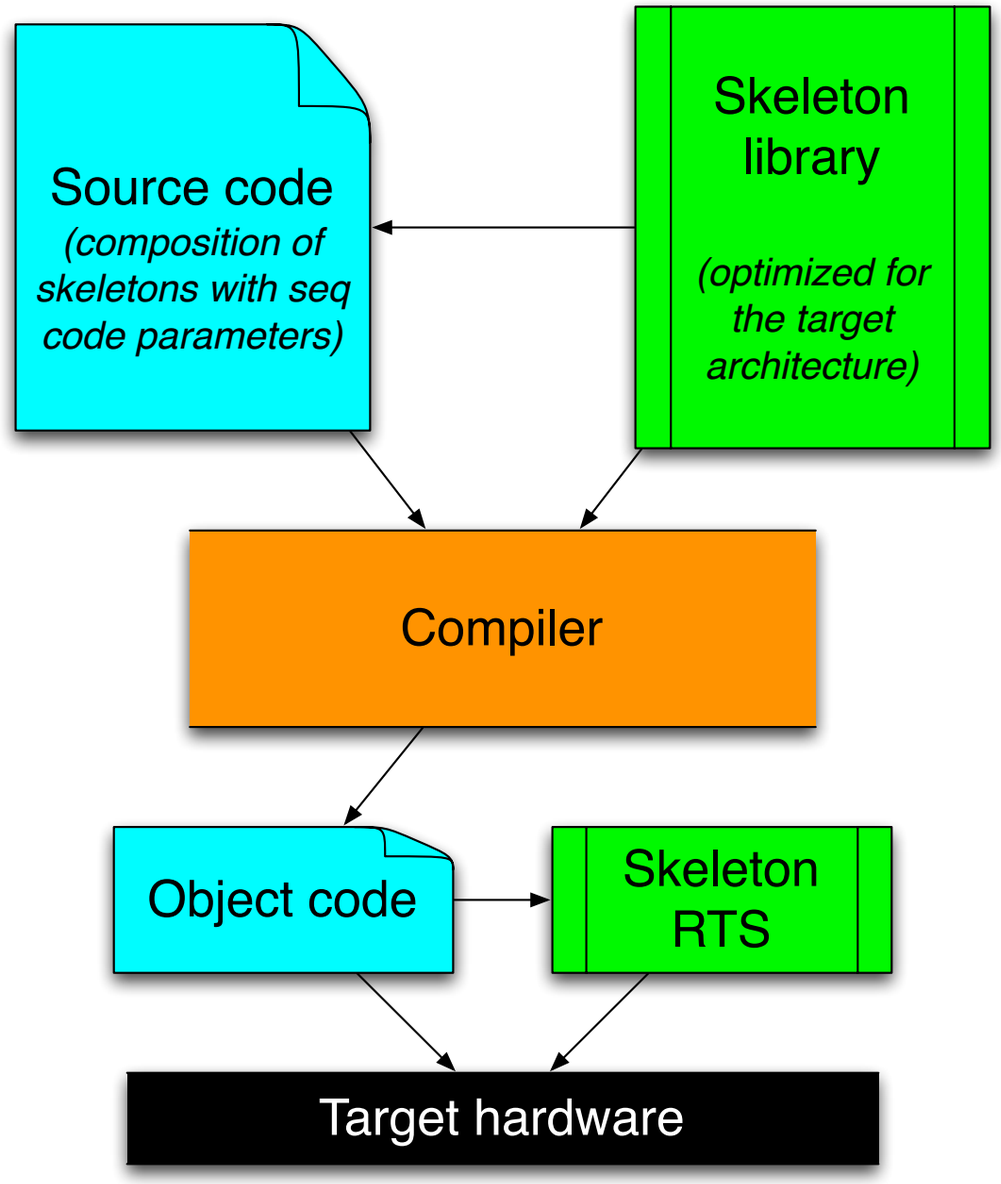
- Know, efficient, reusable, parametric parallelism exploitation pattern
- Skeleton framework
  - *application programmers use skeletons, system designers implement them (separation of concerns)*
  - *nesting (often) allowed (two tier model)*
  - *portability across target architectures*
  - *performance portability*



Skeleton tree

Skeletons @ work

- Application programmer uses available skeletons
  - *specialized through application code*
- then
  - *compiles*
  - *and runs the program*
- no need for target hw expertise



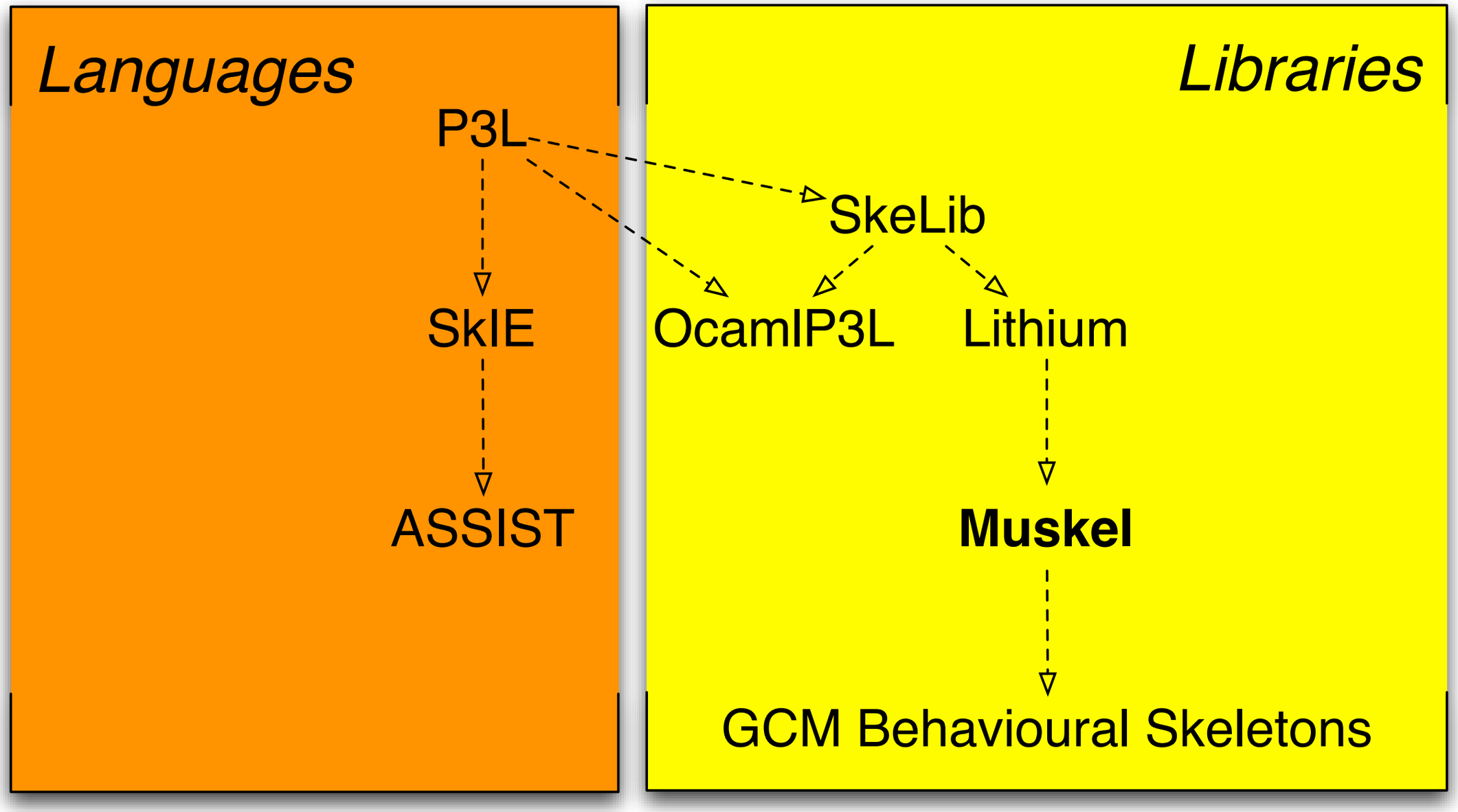
History

late '80
Cole PhD thesis, introduction of the concept

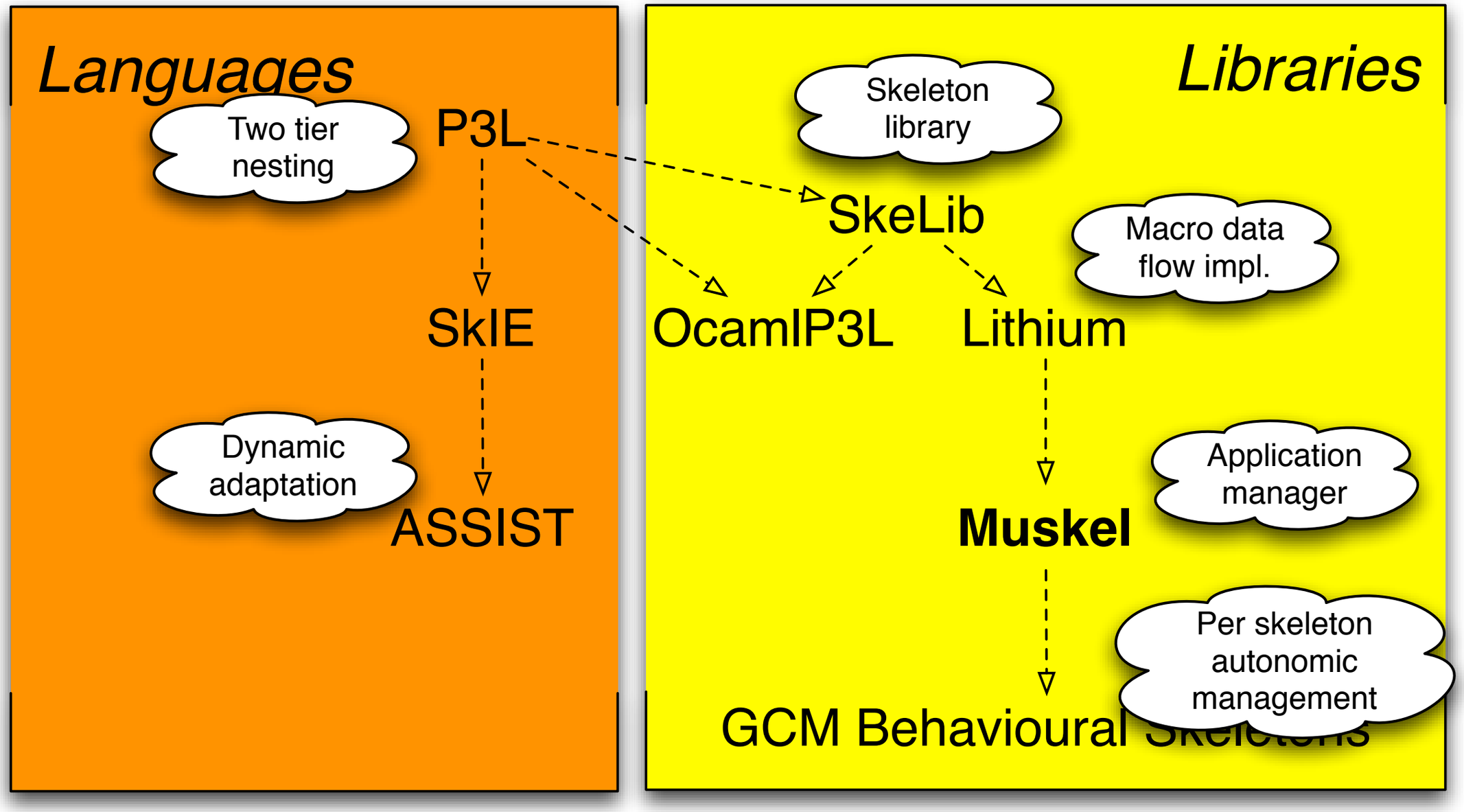
'90
First skeleton frameworks
(P3L Pisa, Imperial College, Skil Muenster)

'00
Assessed skeleton frameworks
eSkel Edinburg, Muesli Muenster, SkeTo
Tokio, ASSIST Pisa, Muskel Pisa, Calcium
Sophia Antipolis, Mallba La Laguna

History


Languages
P3L
SkIE
ASSIST
Libraries
SkeLib
OcamlP3L
Lithium
Muskel
GCM Behavioural Skeletons

History


Languages
Two tier nesting
P3L
SkIE
Dynamic adaptation
ASSIST
Libraries
Skeleton library
SkeLib
Macro data flow impl.
OcamlP3L
Lithium
Muskel
Application manager
Per skeleton autonomic management
GCM Behavioural Skeletons

Muskel

- Macro data flow implementation model (1999)
- Stream parallel skeletons (pipe, farm)
  - *map recently added*
- Application manager
  - *with user supplied contracts and autonomic behaviour*
- User defined skeletons
  - *same efficiency of system provided skeletons*

Muskel (user view)

- *Skeleton* function wrapper
  - *Object compute(Object)*
- Skeletons → program
  - *Farm(Skeleton w), Pipe(Skeleton a, Skeleton b)*
- Manager
  - *params: I/O, program, contract, parallel execution*
- User
  - *defines skeleton programs*
  - *invokes execution through Manager*

Muskel (sample code)

```
import muskel.*;
public class MySkelProgram {
  public static void main(String [] args) {

    Skeleton stage1 = new Stage1();
    Skeleton worker = new Worker();
    Skeleton stage2 = new Farm(worker);
    Skeleton main   = new Pipeline(stage1,stage2);

    InputManager inputStream = new InputStreamMgr(args[0]);
    OutputManager outputStream = new ConsoleOutMgr();

    PerfContract parDegree =
      new ParDegree(Integer.parseInt(args[1]);

    Manager mgr =
      new Manager(main,inputStream,outputStream,parDegree);

    mgr.compute();
  }
}
```

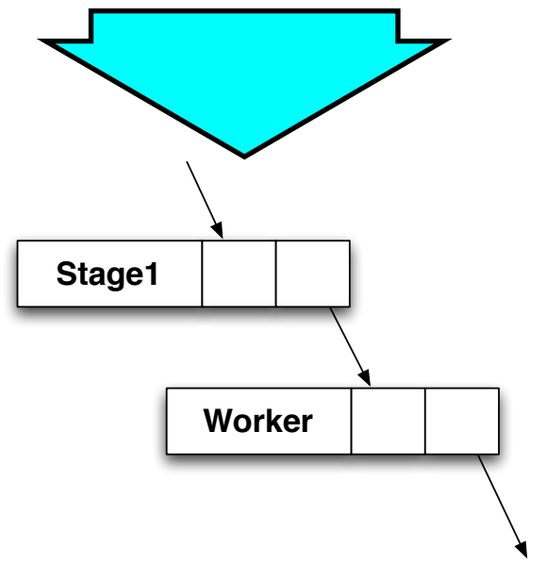
Muskel (macro data flow)

- Skeleton program ➙ macro data flow graph
- Input task ➙ MDF graph instance
- Execution ➙ distributed macro data flow interpreter
  - *get fireable instruction (from any graph) ➙ execute ➙ store back result tokens ➙ loop*

Muskel (macro data flow)

- Skeleton program ➙ macro data flow graph
- Input task ➙ MDF graph instance
- Execution ➙ distributed macro data flow interpreter
  - *get fireable instruction (from any graph) ➙ execute ➙ store back result tokens ➙ loop*

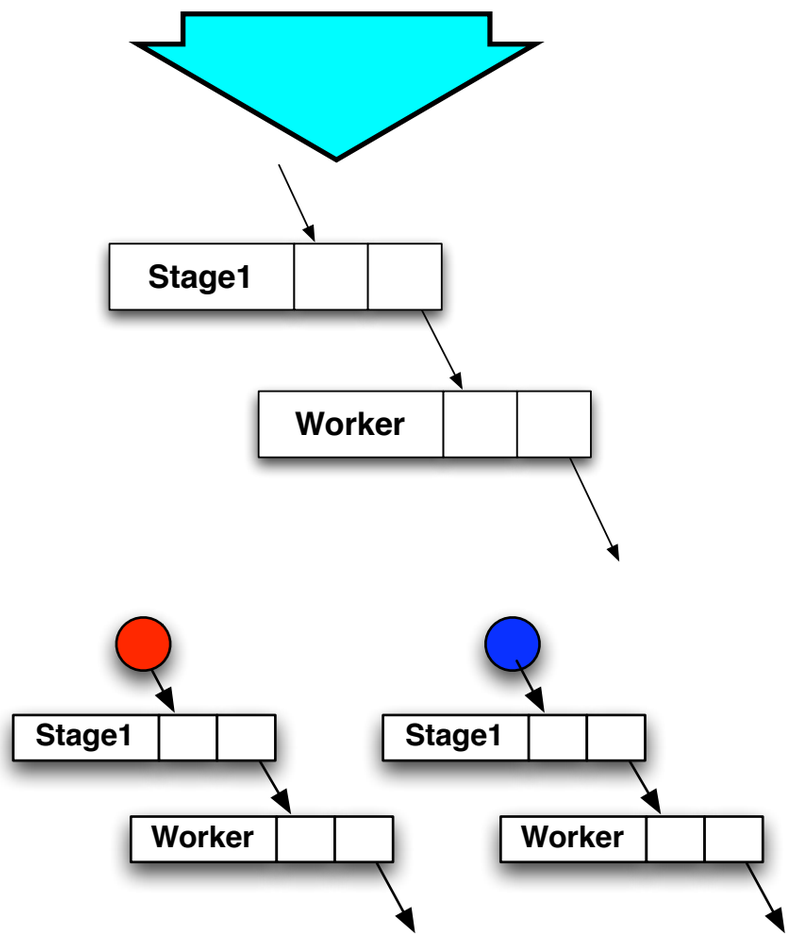
```
Skeleton stage1 =
  new Stage1();
Skeleton worker =
  new Worker();
Skeleton stage2 =
  new Farm(worker);
Skeleton main   =
  new Pipeline(stage1,stage2);
```



Muskel (macro data flow)

- Skeleton program ➙ macro data flow graph
- Input task ➙ MDF graph instance
- Execution ➙ distributed macro data flow interpreter
  - *get fireable instruction (from any graph) ➙ execute ➙ store back result tokens ➙ loop*

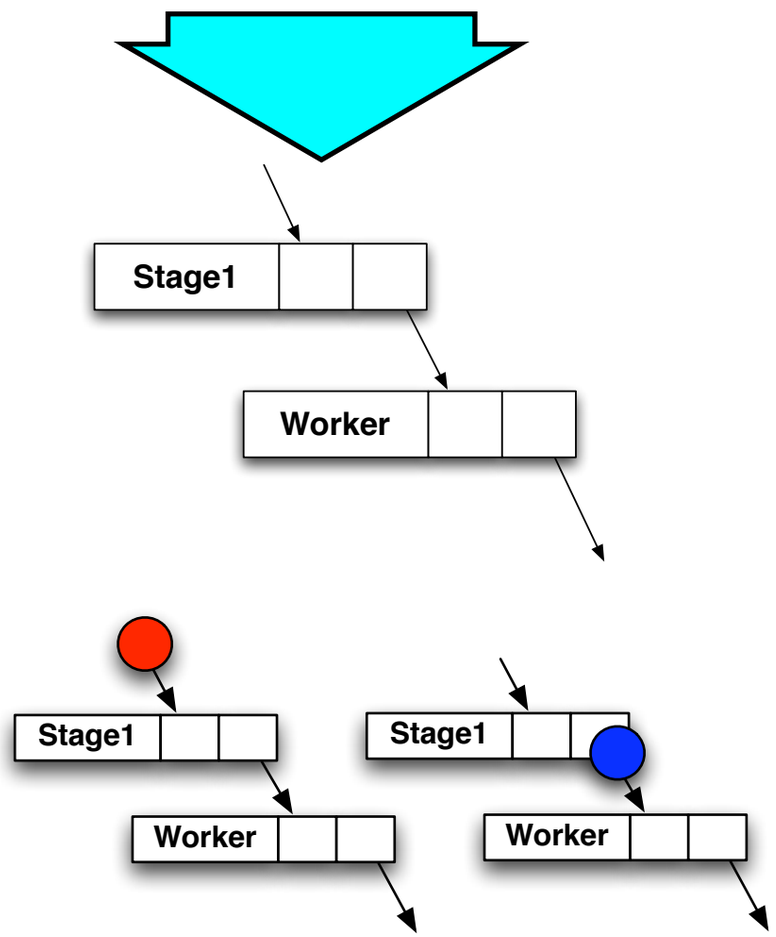
```
Skeleton stage1 =
  new Stage1();
Skeleton worker =
  new Worker();
Skeleton stage2 =
  new Farm(worker);
Skeleton main   =
  new Pipeline(stage1,stage2);
```



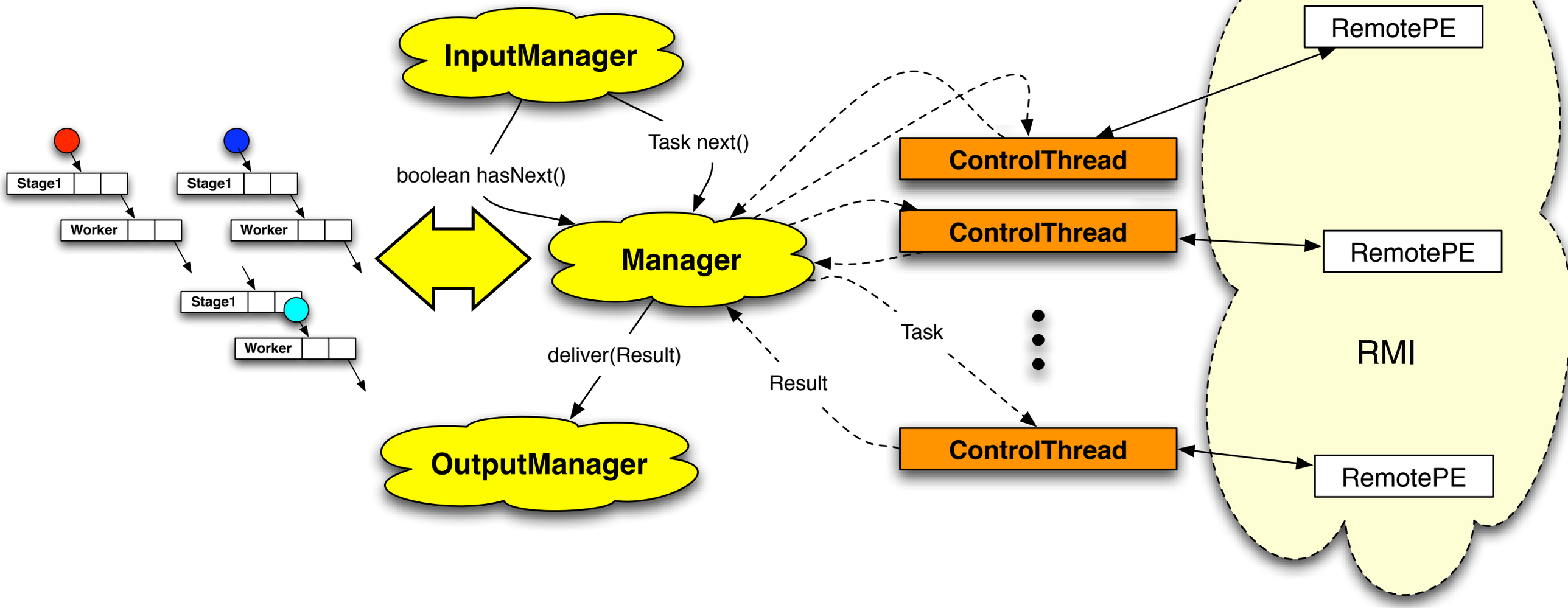
Muskel (macro data flow)

- Skeleton program ➙ macro data flow graph
- Input task ➙ MDF graph instance
- Execution ➙ distributed macro data flow interpreter
  - *get fireable instruction (from any graph) ➙ execute ➙ store back result tokens ➙ loop*

```
Skeleton stage1 =
  new Stage1();
Skeleton worker =
  new Worker();
Skeleton stage2 =
  new Farm(worker);
Skeleton main   =
  new Pipeline(stage1,stage2);
```



Muskel (execution)



Muskel: Orc modeling

$$system(pgm, tasks, contract, G, t) \triangleq$$
$$taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t)$$

$$discovery(G, pgm, t) \triangleq$$
$$(|_{g \in G} \ (\ \text{if } remw \neq \text{false } \gg rworkerpool.add(remw)$$
$$\text{where } remw :\in (\ g.can_execute(pgm) \mid Rtimer(t) \gg let(false)\ )\ )\ )$$
$$\gg discovery(G, pgm, t)$$

$$manager(pgm, contract, t) \triangleq$$
$$|_{i\ 1 \leq i \leq contract:} (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t))$$
$$\mid monitor$$

$$ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk >$$
$$(\ \text{if } valid \gg resultpool.add(r) \gg ctrlthread_i(pgm, remw, t)$$
$$\mid \text{if } \neg valid \gg (\ taskpool.add(tk)$$
$$\mid alarm.put(i) \gg c_i.get > w > ctrlthread_i(pgm, w, t)\ )\ )$$
$$\text{where } (valid, r) :\in$$
$$(\ remw(pgm, tk) > r > let(true, r) \mid Rtimer(t) \gg let(false, 0)\ )$$

$$monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw)$$
$$\gg monitor$$

Muskel: Orc modeling

3 parallel activities in the system: task repository, discovery and manager

$$system(pgm, tasks, contract, G, t) \triangleq taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t)$$

$$discovery(G, pgm, t) \triangleq (|_{g \in G} \ (\ \text{if } remw \neq \text{false} \gg rworkerpool.add(remw) \ \text{where } remw :\in (\ g.can_execute(pgm) \mid Rtimer(t) \gg let(false)\ )\ )\ ) \gg discovery(G, pgm, t)$$

$$manager(pgm, contract, t) \triangleq |_{i\ 1 \leq i \leq contract:} (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t)) \mid monitor$$

$$ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk > (\ \text{if } valid \gg resultpool.add(r) \gg ctrlthread_i(pgm, remw, t) \mid \text{if } \neg valid \gg (\ taskpool.add(tk) \mid alarm.put(i) \gg c_i.get > w > ctrlthread_i(pgm, w, t)\ )\ )$$
$$\text{where } (valid, r) :\in (\ remw(pgm, tk) > r > let(true, r) \mid Rtimer(t) \gg let(false, 0)\ )$$

$$monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw) \gg monitor$$

Muskel: Orc modeling

3 parallel activities in the system: task repository, discovery and manager

discovery adds resources to the resource repository (infinite loop)

$$system(pgm, tasks, contract, G, t) \triangleq$$
$$taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t)$$

$$discovery(G, pgm, t) \triangleq$$
$$(|_{g \in G} \ (\ \text{if } remw \neq \text{false} \gg rworkerpool.add(remw)$$
$$\text{where } remw :\in (\ g.can_execute(pgm) \mid Rtimer(t) \gg let(false)\ )\ )\ )$$
$$\gg discovery(G, pgm, t)$$

$$manager(pgm, contract, t) \triangleq$$
$$|_{i\ 1 \leq i \leq contract:} (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t))$$
$$\mid monitor$$

$$ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk >$$
$$(\ \text{if } valid \gg resultpool.add(r) \gg ctrlthread_i(pgm, remw, t)$$
$$\mid \text{if } \neg valid \gg (\ taskpool.add(tk)$$
$$\mid alarm.put(i) \gg c_i.get > w > ctrlthread_i(pgm, w, t)\ )\ )$$
$$\text{where } (valid, r) :\in$$
$$(\ remw(pgm, tk) > r > let(true, r) \mid Rtimer(t) \gg let(false, 0)\ )$$

$$monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw)$$
$$\gg monitor$$

Muskel: Orc modeling

3 parallel activities in the system: task repository, discovery and manager

discovery adds resources to the resource repository (infinite loop)

manager forks control threads (user required #)

$$system(pgm, tasks, contract, G, t) \triangleq$$
$$taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t)$$

$$discovery(G, pgm, t) \triangleq$$
$$(|_{g \in G} \ (\ \text{if } remw \neq \text{false } \gg rworkerpool.add(remw)$$
$$\text{where } remw :\in (\ g.can_execute(pgm) \mid Rtimer(t) \gg let(false)\ )\ )\ )$$
$$\gg discovery(G, pgm, t)$$

$$manager(pgm, contract, t) \triangleq$$
$$|_{i\ 1 \leq i \leq contract:} (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t))$$
$$\mid monitor$$

$$ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk >$$
$$(\ \ \text{if } valid \gg resultpool.add(r) \gg ctrlthread_i(pgm, remw, t)$$
$$\mid \text{if } \neg valid \gg (\ taskpool.add(tk)$$
$$\mid alarm.put(i) \gg c_i.get > w > ctrlthread_i(pgm, w, t)\ )\ )$$
$$\text{where } (valid, r) :\in$$
$$(\ \ remw(pgm, tk) > r > let(true, r) \mid Rtimer(t) \gg let(false, 0)\ \ )$$

$$monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw)$$
$$\gg monitor$$

Muskel: Orc modeling

3 parallel activities in the system: task repository, discovery and manager

discovery adds resources to the resource repository (infinite loop)

manager forks control threads (user required #)

a control thread looks for a fireable MDF instr, executes it remotely and in case of fault it stores back the instruction and a new control thread is forked

$$system(pgm, tasks, contract, G, t) \triangleq$$
$$taskpool.add(tasks) \mid discovery(G, pgm, t) \mid manager(pgm, contract, t)$$

$$discovery(G, pgm, t) \triangleq$$
$$(|_{g \in G} \ (\ \text{if } remw \neq \text{false} \gg rworkerpool.add(remw)$$
$$\text{where } remw :\in (\ g.can_execute(pgm) \mid Rtimer(t) \gg let(false)\ )\ )\ )$$
$$\gg discovery(G, pgm, t)$$

$$manager(pgm, contract, t) \triangleq$$
$$|_{i\ 1 \leq i \leq contract:} (rworkerpool.get > remw > ctrlthread_i(pgm, remw, t))$$
$$\mid monitor$$

$$ctrlthread_i(pgm, remw, t) \triangleq taskpool.get > tk >$$
$$(\ \text{if } valid \gg resultpool.add(r) \gg ctrlthread_i(pgm, remw, t)$$
$$\mid \text{if } \neg valid \gg (\ taskpool.add(tk)$$
$$\mid alarm.put(i) \gg c_i.get > w > ctrlthread_i(pgm, w, t)\ )\ )$$
$$\text{where } (valid, r) :\in$$
$$(\ remw(pgm, tk) > r > let(true, r) \mid Rtimer(t) \gg let(false, 0)\ )$$

$$monitor \triangleq alarm.get > i > rworkerpool.get > remw > c_i.put(remw)$$
$$\gg monitor$$

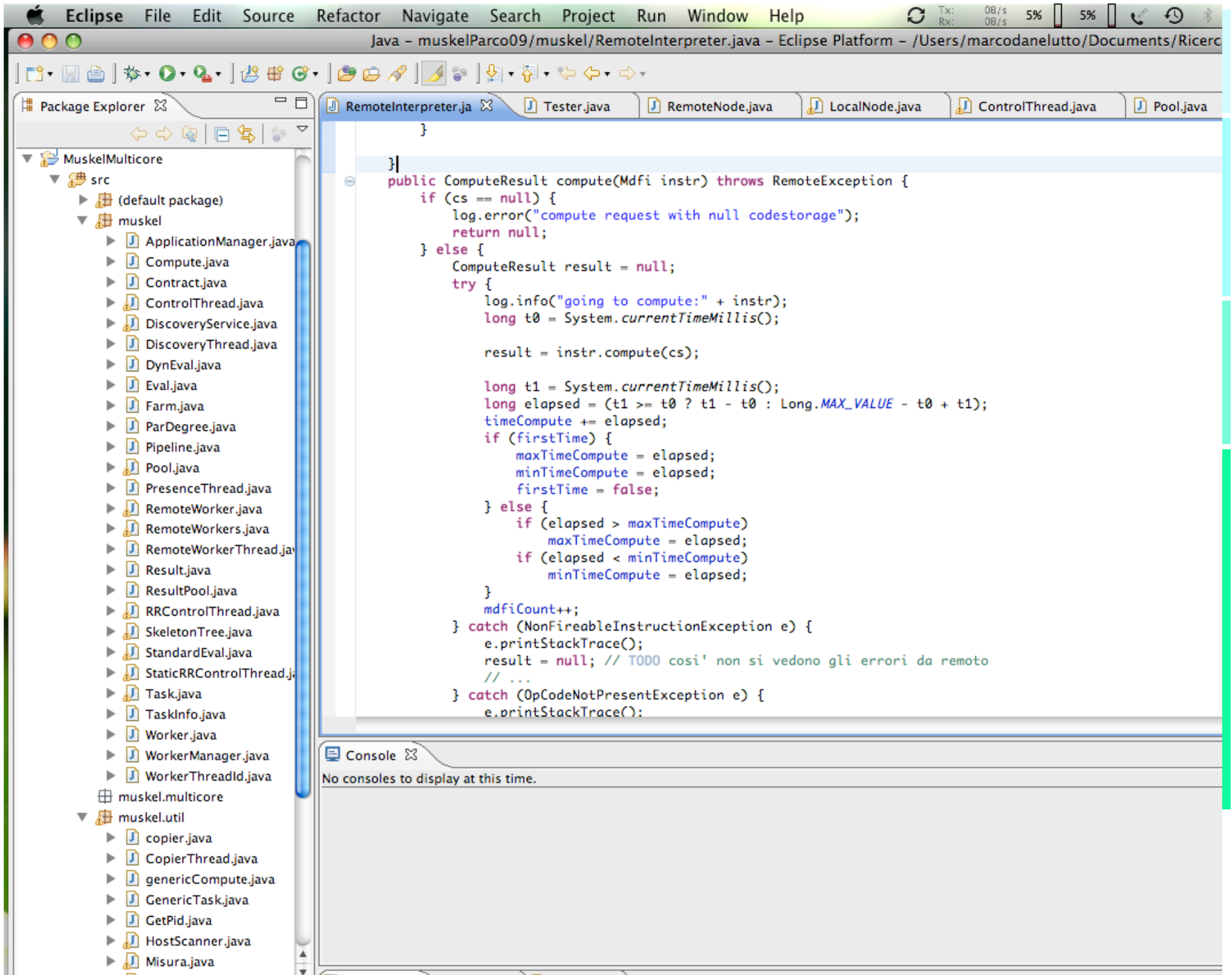
Muskel: Orc modeling

3 parallel activities in the system: task repository, discovery and manager

discovery adds resources to the resource repository (infinite loop)

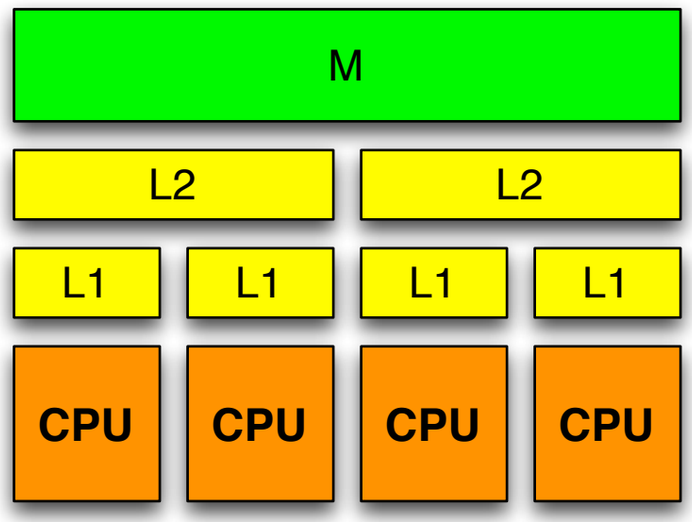
manager forks control threads (user required #)

a control thread looks for a fireable MDF instr, executes it remotely and in case of fault it stores back the instruction and a new control thread is forked



Muskel going MC

- Multi-core
  - *single chip, shared memory hierarchy, private caches*
- Single multi-core
  - *minor changes:*
    - control thread operates on RemoteInterpreter Objects rather than looking up RMI objects
    - #core discovery through /proc inspection



Modeling

- The original discovery:

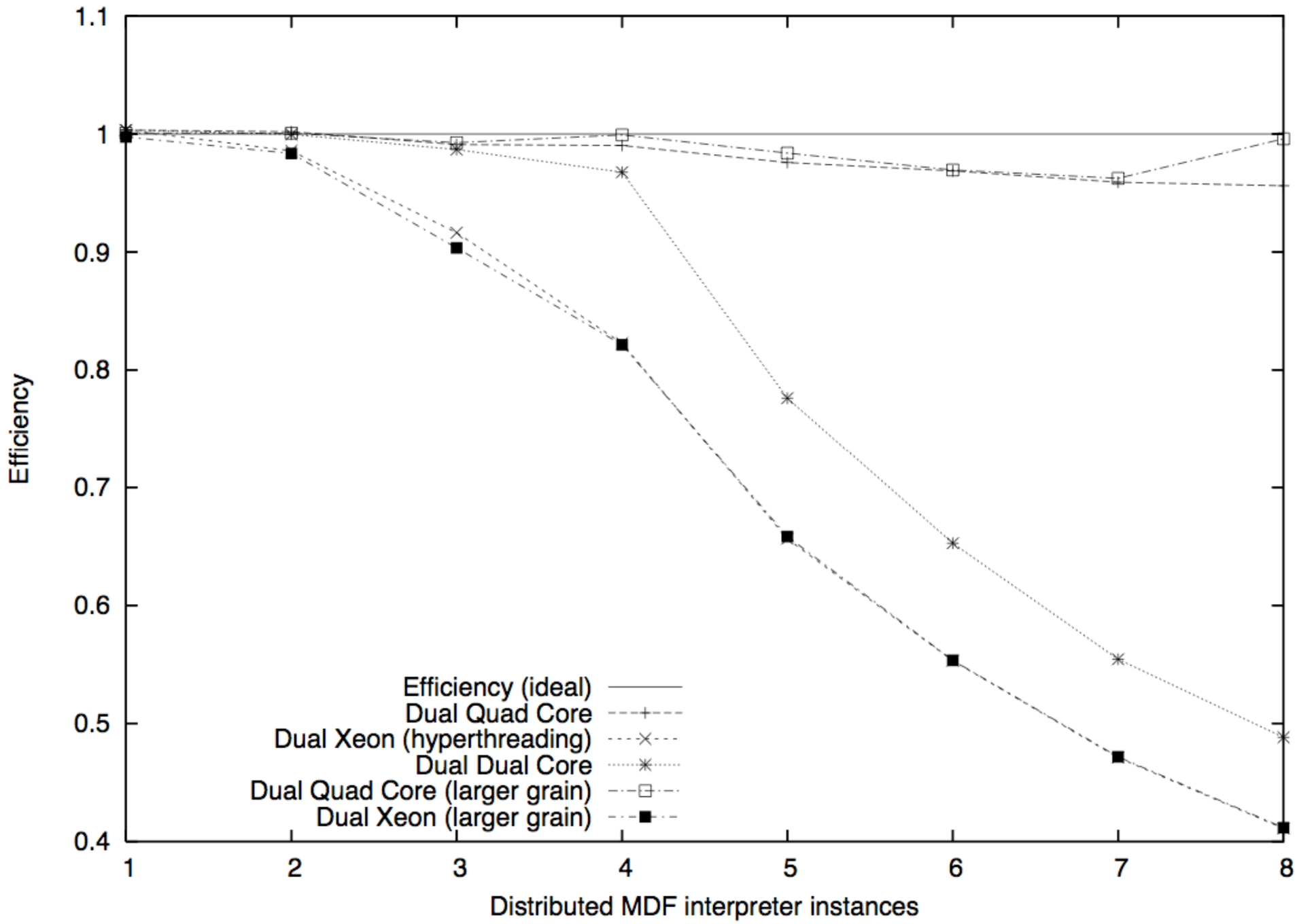
$$discovery(G, pgm, t) \triangleq$$
$$(|_{g \in G} \text{ (if } remw \neq \text{false } \gg rworkerpool.add(remw)$$
$$\text{where } remw :\in (\ g.can_execute(pgm)\ \mid Rtimer(t) \gg let(false)\ )\ )$$
$$\gg discovery(G, pgm, t)$$

- is substituted by:

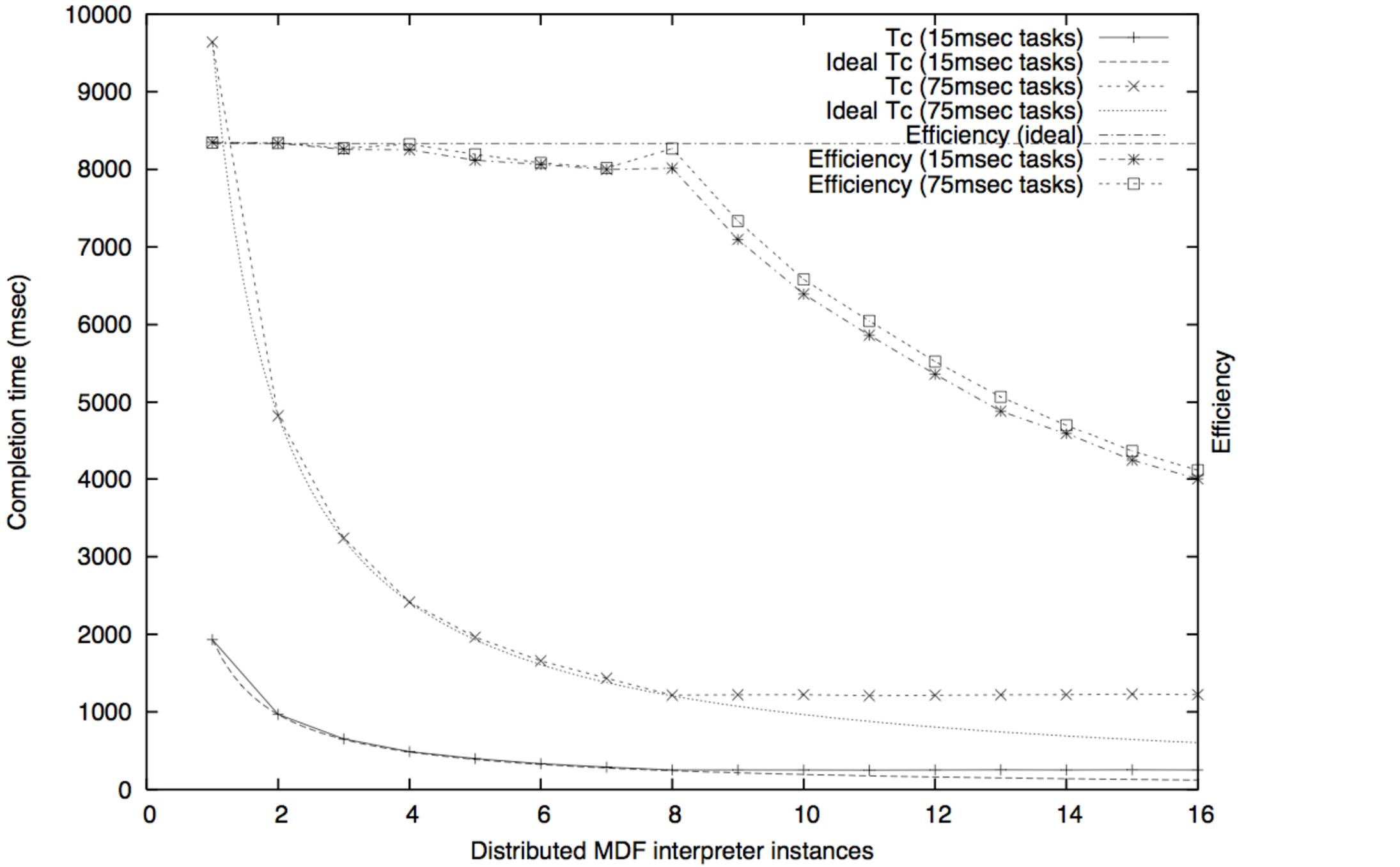
$$discovery(G, pgm, t) \triangleq$$
$$(|_{i=1,n_{core}} let(worker.new())) > w > rmworkerpool.add(w)$$

- an object (rather than a remote object) is eventually used by control threads

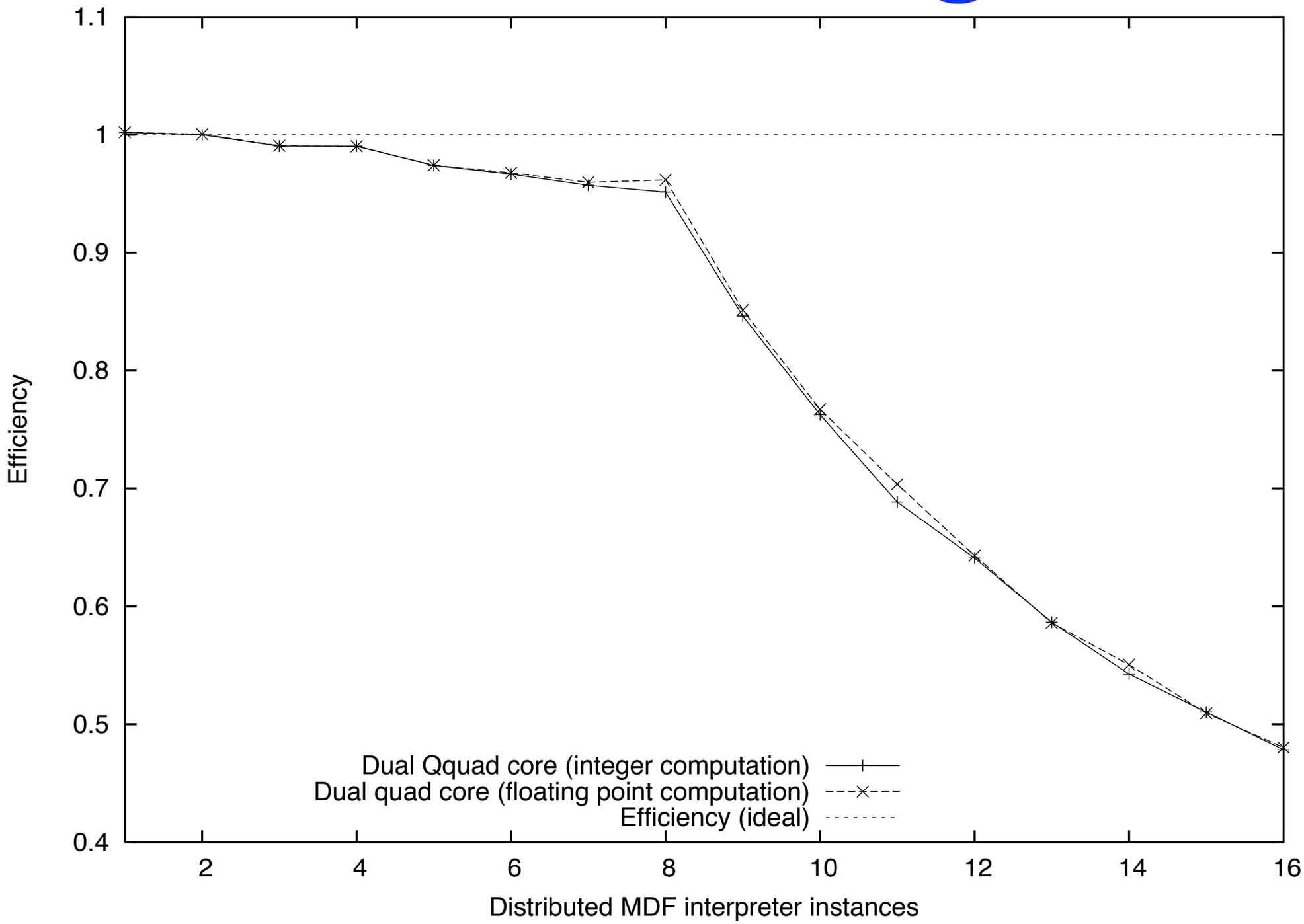
Scalability



Grain


Completion time (msec)
Efficiency
Distributed MDF interpreter instances
Tc (15msec tasks)
Ideal Tc (15msec tasks)
Tc (75msec tasks)
Ideal Tc (75msec tasks)
Efficiency (ideal)
Efficiency (15msec tasks)
Efficiency (75msec tasks)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
0
2
4
6
8
10
12
14
16

Integer vs. FP



Muskel going MC-NOW

- Discovery process modified:
  - *PEs announce IP, MDF interpreter version,* ***core #***
- Manager modified:
  - ***cores*** *recruited up to ParDegree*
    - several policies used (e.g. all cores first, round robin on MC-PEs, ...)
- RMI native multi threading exploited on Remote Interpreters (stateless!)

Modeling

- Original discovery process

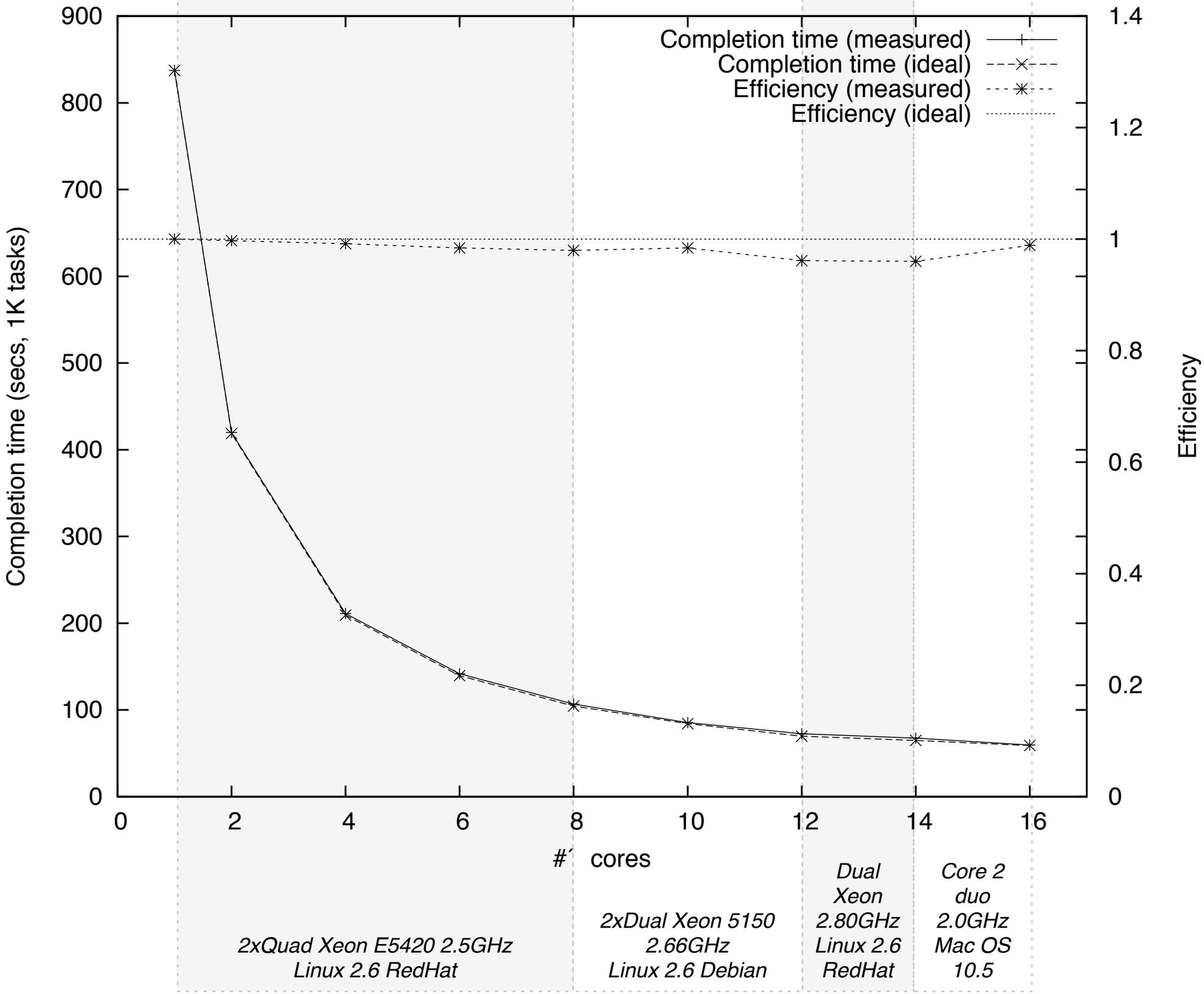
$$|_{g \in G}\ (\ \text{if } remw \neq \text{false } \gg rworkerpool.add(remw)$$
$$\text{where } remw :\in (g.can_execute(pgm) | Rtimer(t) \gg let(false))\ )$$

- is changed to:

$$(|_{g \in G}\ (\ \text{if } remw \neq \text{false} \gg (|_{i \in [1,\#ncore]}\ rworkerpool.add(remw)\ )$$
$$\text{where } (remw, \#ncore) :\in$$
$$(g.can_execute(pgm) | Rtimer(t) \gg let(false))\ )\ )$$

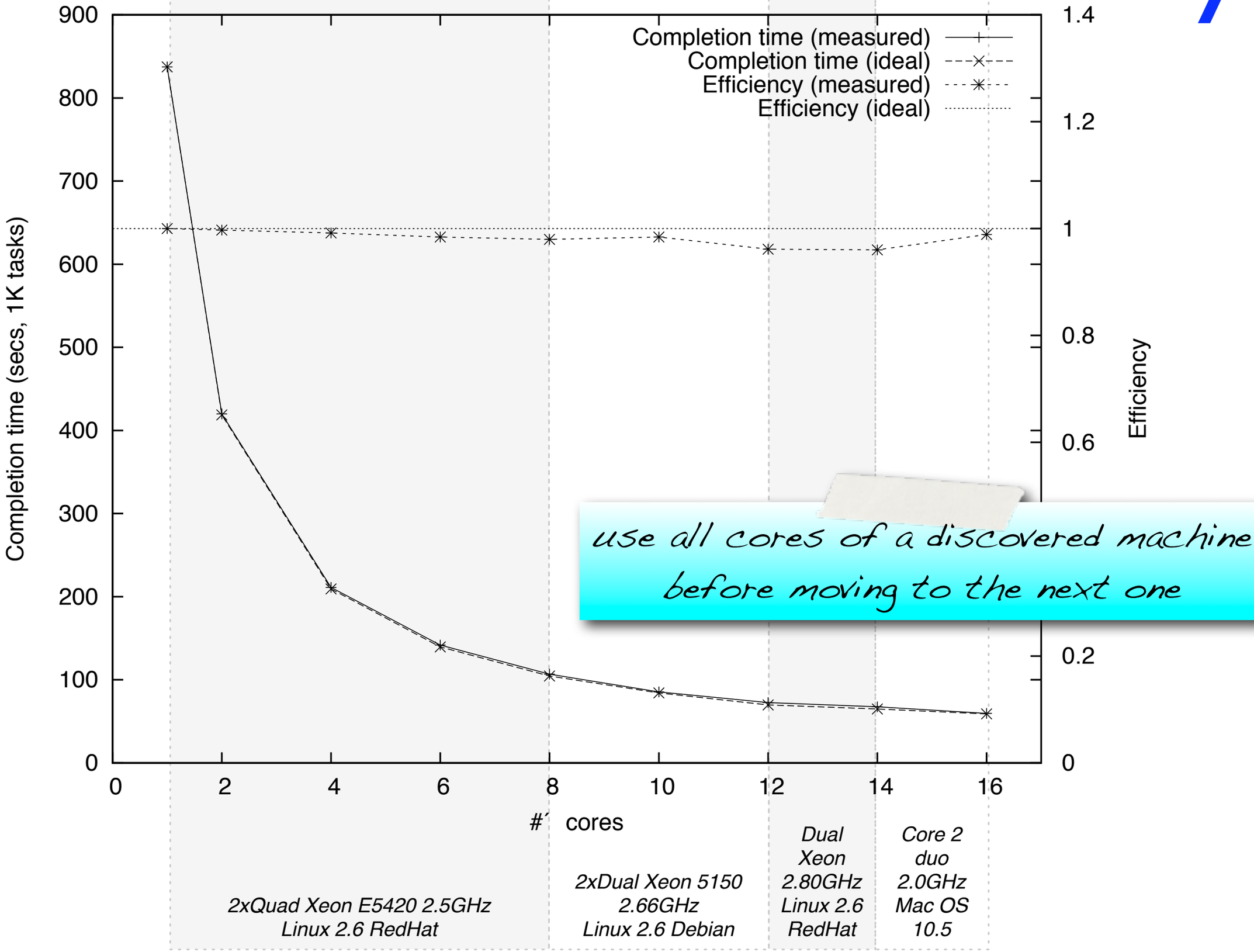
- core number of remote nodes is reported to the manager

MC-NOW scalability



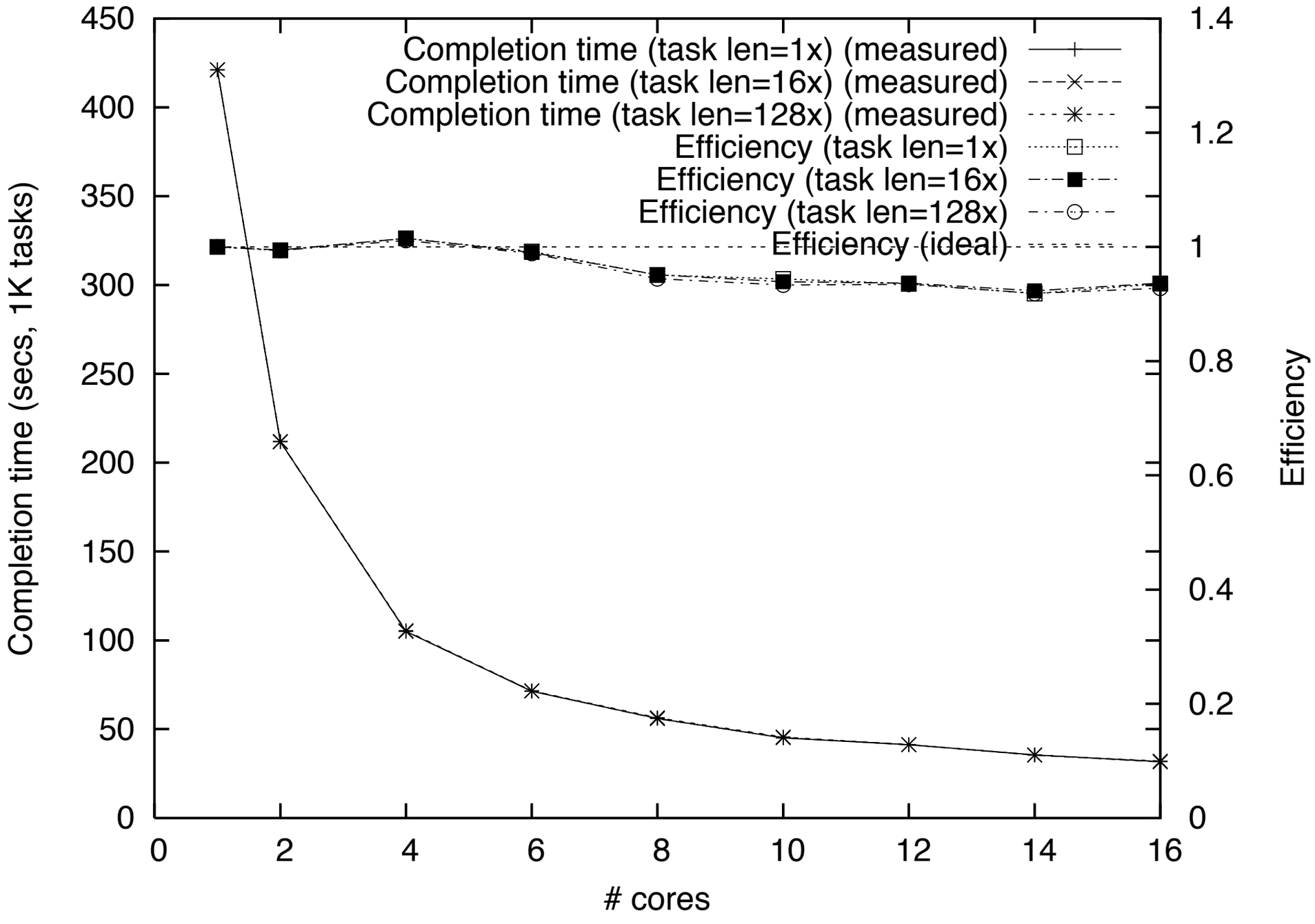
CoreGRID
ERCIM

MC-NOW scalability

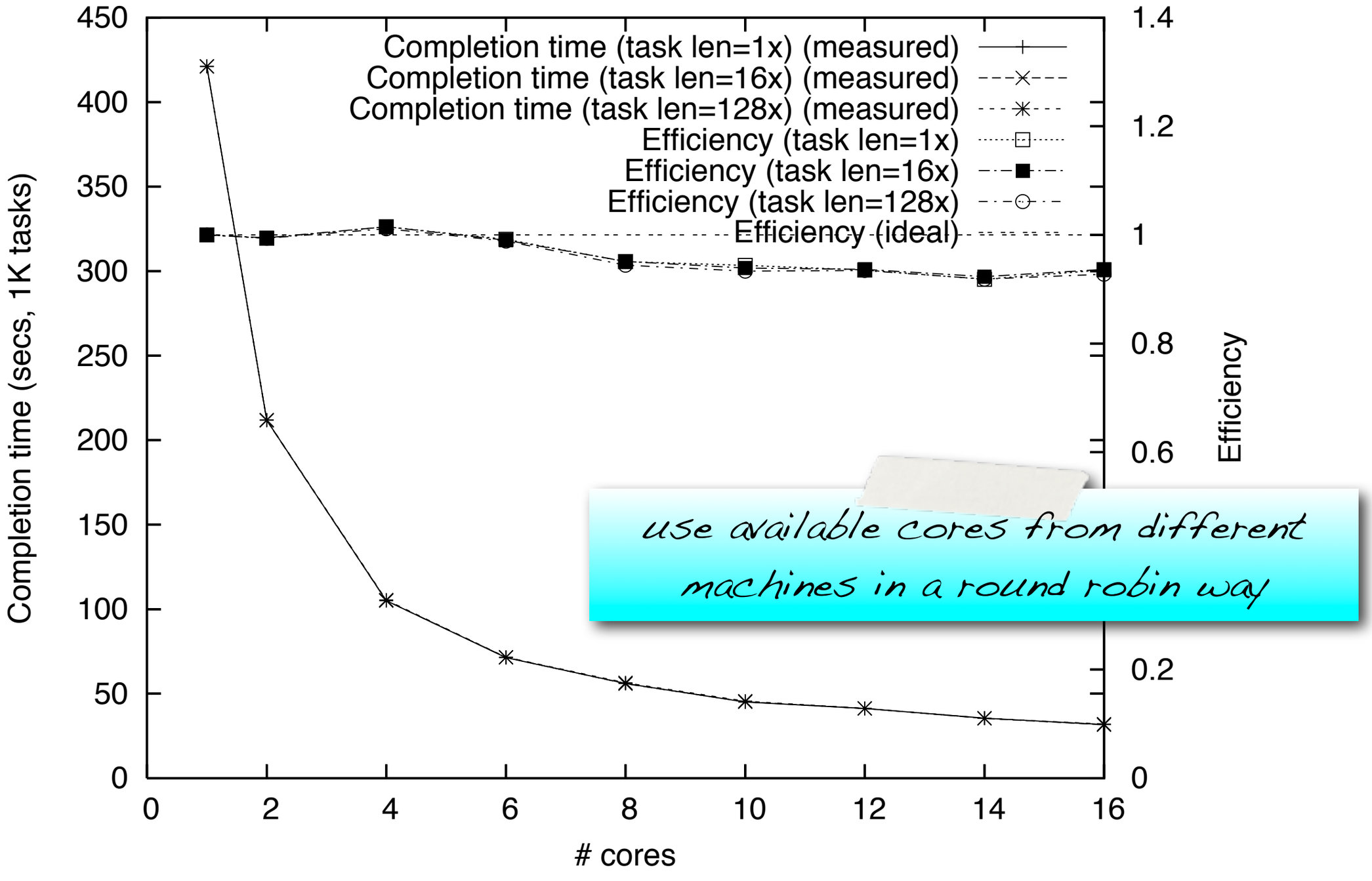


use all cores of a discovered machine before moving to the next one

MC-NOW scalability



MC-NOW scalability



use available cores from different machines in a round robin way

Why does it work ?

- Medium/Coarse grain
  - *tens of milliseconds of computation per in/out packet*
  - *control threads do not compete too much to access task pool*
- Communication/serialization hiding
  - *parallelism "in eccess"*
- Skeleton structure
  - *much more knowledge available than with traditional, "free hand" parallel programming*

Conclusions

- Feasibility of porting skeleton technology to multi (many?) core systems
  - *seamless, not the same for other programming models*
- Assessed by experimental results
  - *on single multi core system, on MC COW/NOWs*
- Implementation supported by ORC modeling
  - *significantly simplifies the programming effort*

Future work

- Experimenting different recruitment policies
  - *user configurable through Manager*
- Porting experience to many core processing elements
  - *many core programming model harder than multi core*
  - *but most of the advantages in the muskel approach are preserved*

Thank you for your attention

Any questions?