Second generation skeleton systems

Marco Danelutto
Dept. Computer Science
University of Pisa

Contents

- Skeletons
- The Pisa experience
- Cole's manifesto
- Additions to Cole's manifesto
- **`muskel`**
- ASSIST
- Conclusions

Focus of the talk

- Assume you know skeletons …
- Focus on *skeleton concept evolution*
  - Concept
  - Implementation
  - Results
- Report on Pisa experience
  - Working on skeletons since 1990
  - Several prototypes / environments
  - Several lessons learned (!)

Skeletons

- Known, efficient, common parallelism exploitation patterns
  - AKA: templates, design patterns, coordination patterns, components, …
- Since late '80 (Cole's PhD thesis)
- Range of flavors
  - Languages/libraries
  - Functional/imperative
  - Composable/flat

Evolution

Cole PhD (1988)
Fixed degree DC, Iterative combination, Cluster Task queue

Darlington (1992)
Pipeline, Farm, RaMP, DMPA

SCL Fortran S

P3L (1991)
Pipeline, Farm, Map, Reduce

SkIE ASSIST Lithium OcamlP3L

BMF ('80)
map fold reduce prefix + algebra

La Laguna ('90)
Advance perf modeling

Serot (1999)
Skipper (→MDF)

Gorlatch (late '90)

Skillicorn (mid '90)

Kuchen Skil (1998)

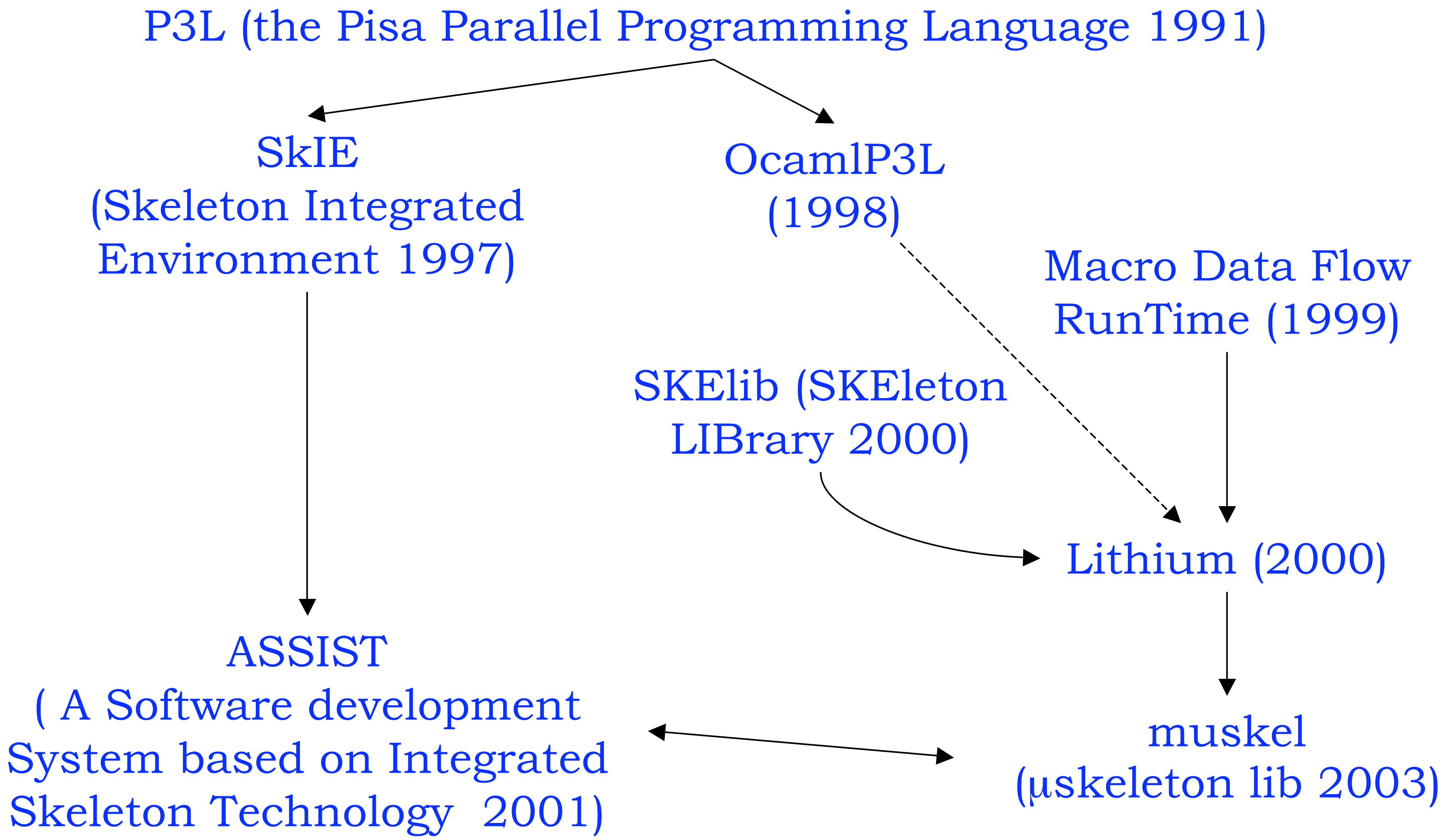
SkeTo (early '00)

HOC (early '00)

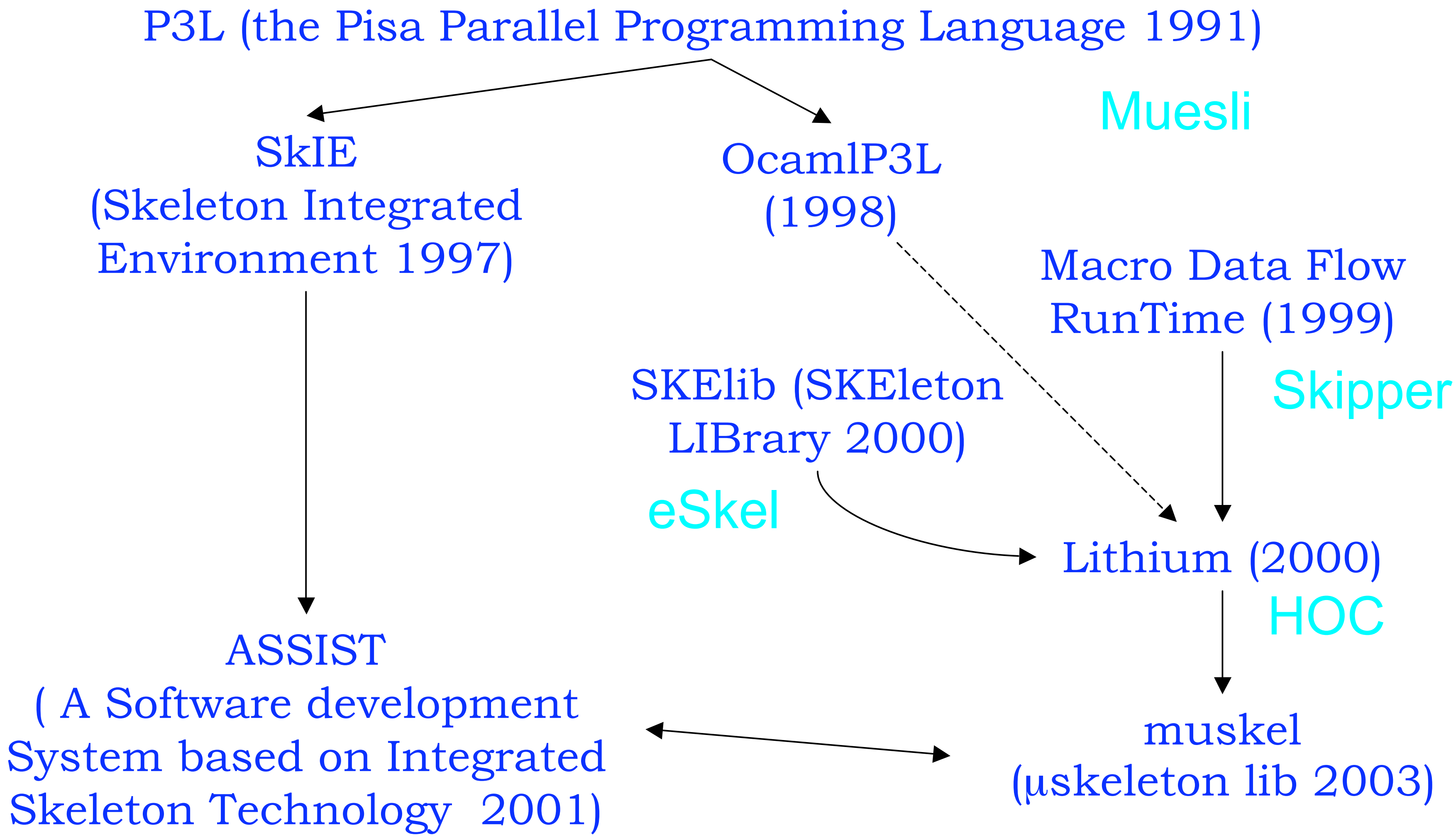
Kuchen Muesli (2002)
Pipeline, Farm, Parallel array + collectives

eSkel (2002)
Parametric skeletons + Give/Take

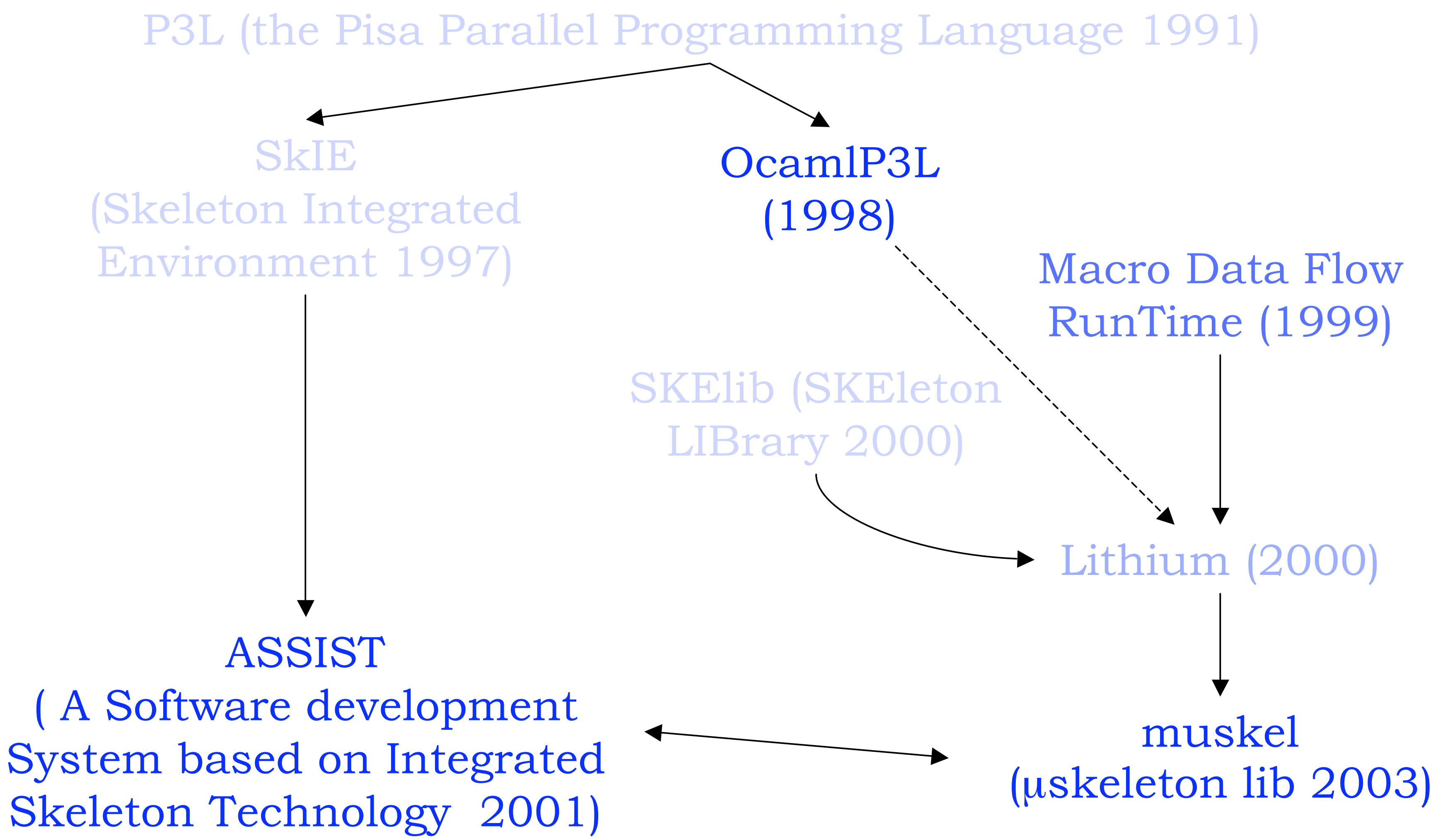
The Pisa picture



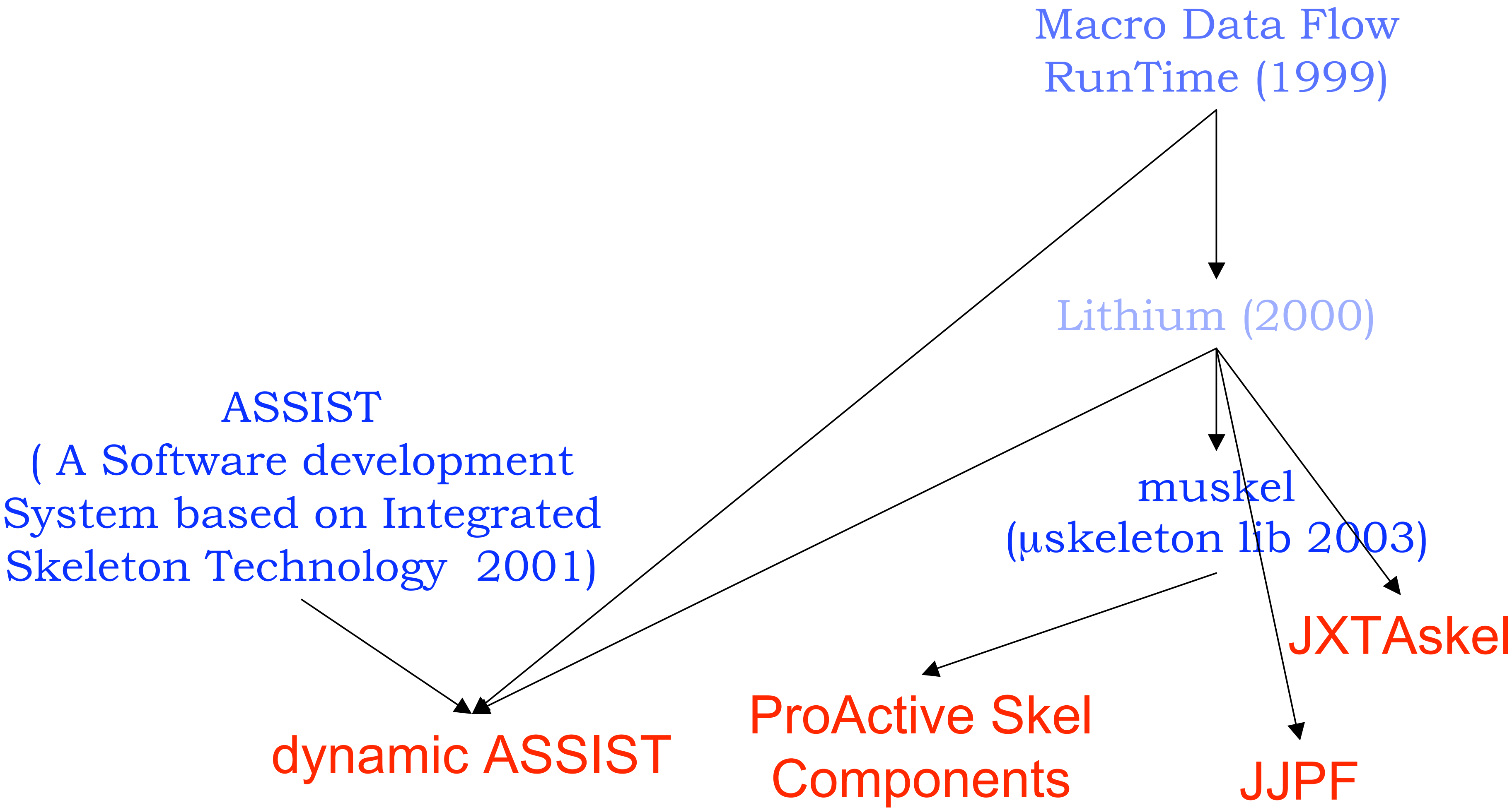
The Pisa picture: influences …



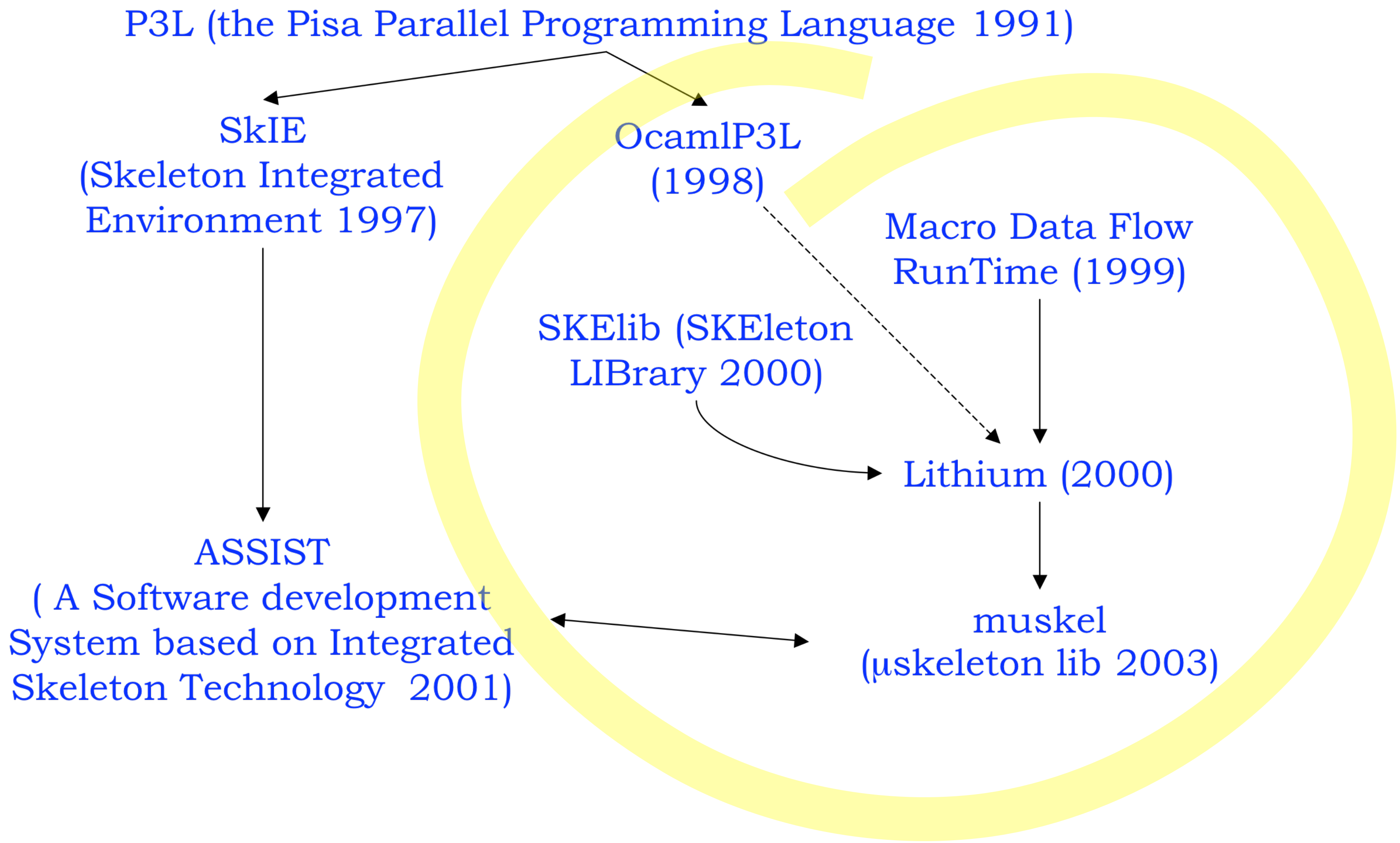
The Pisa picture: alive projects



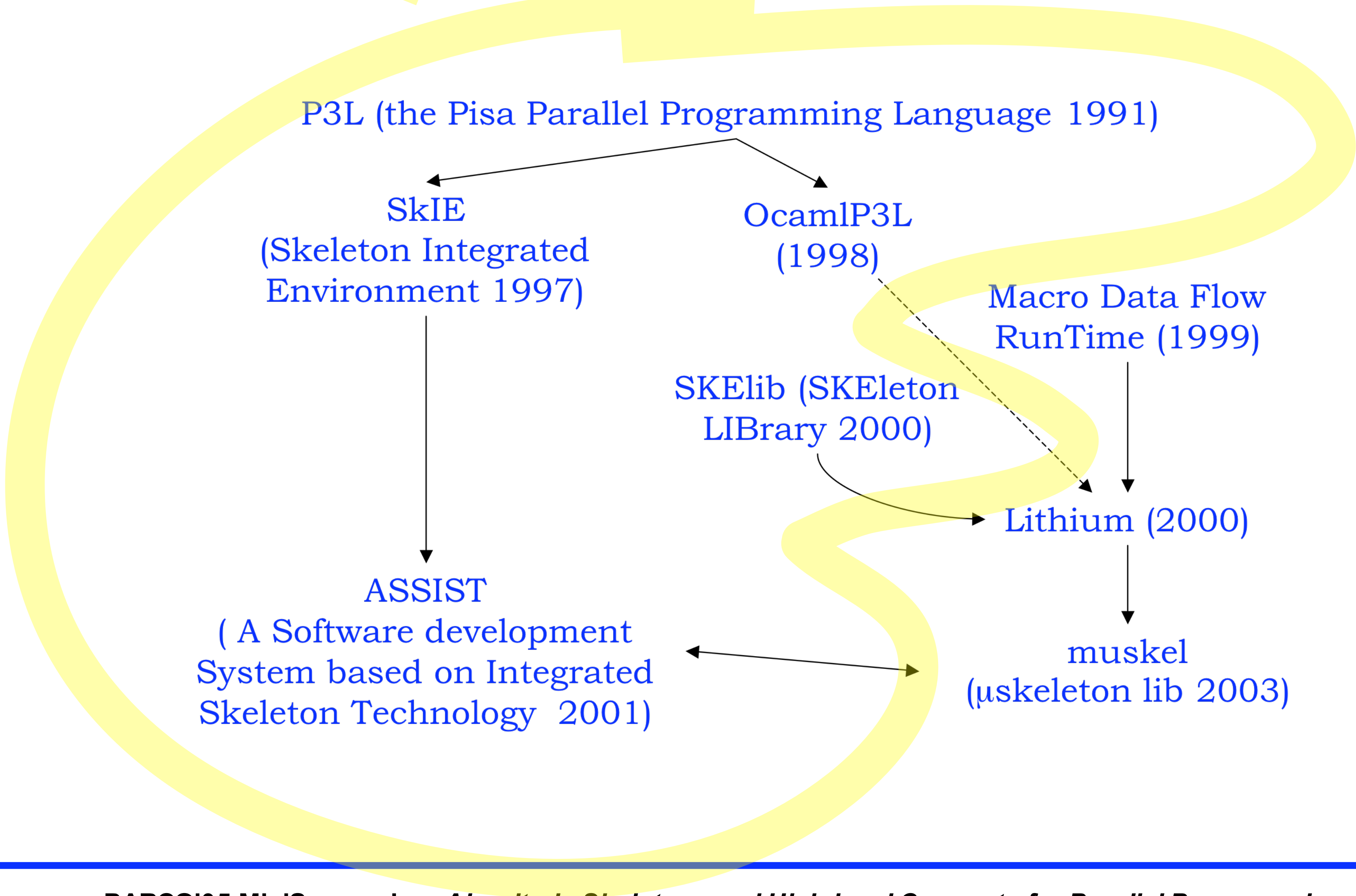
The Pisa picture: side projects



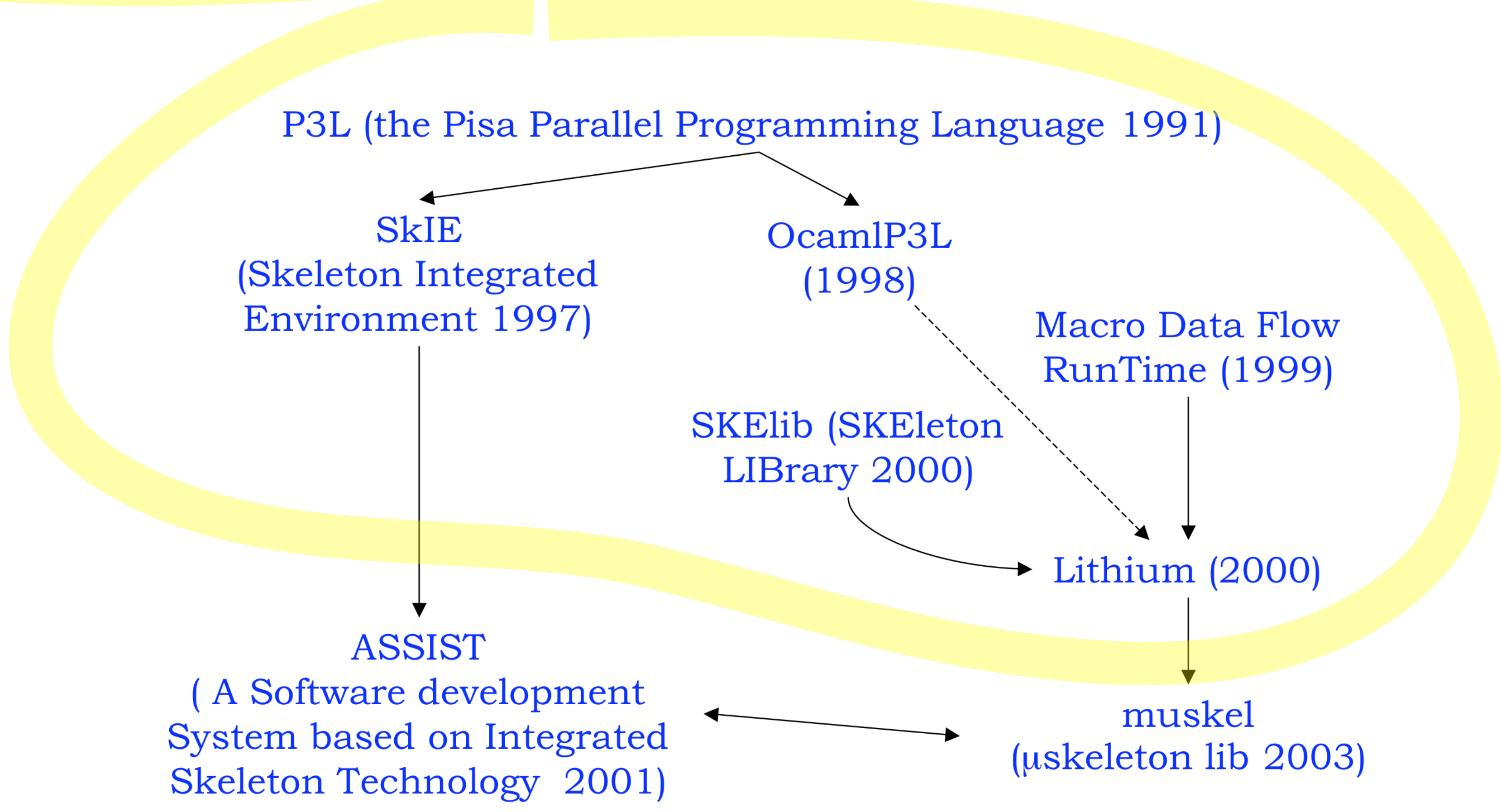
Library vs. Language



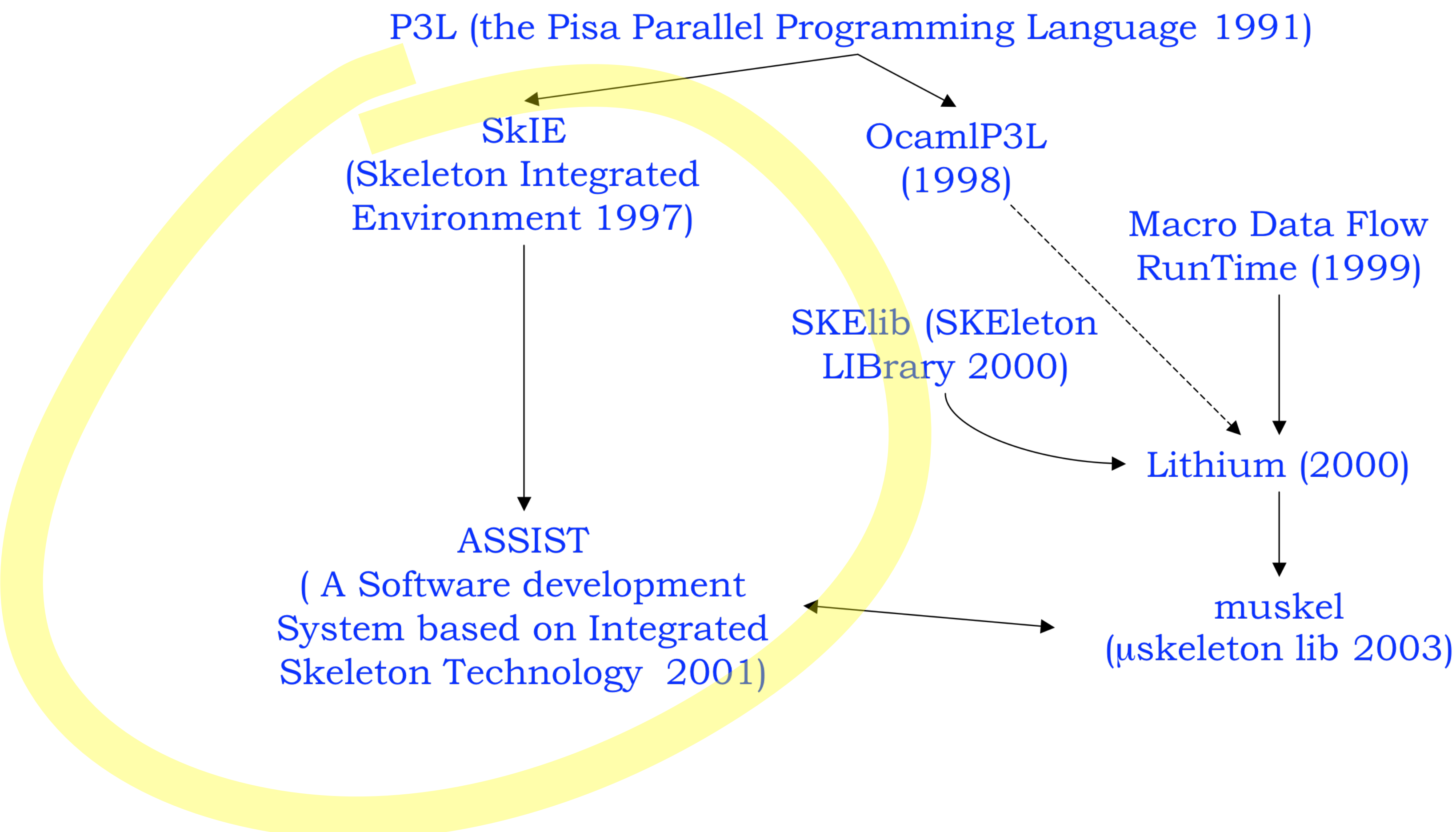
Template vs. Macro Data Flow



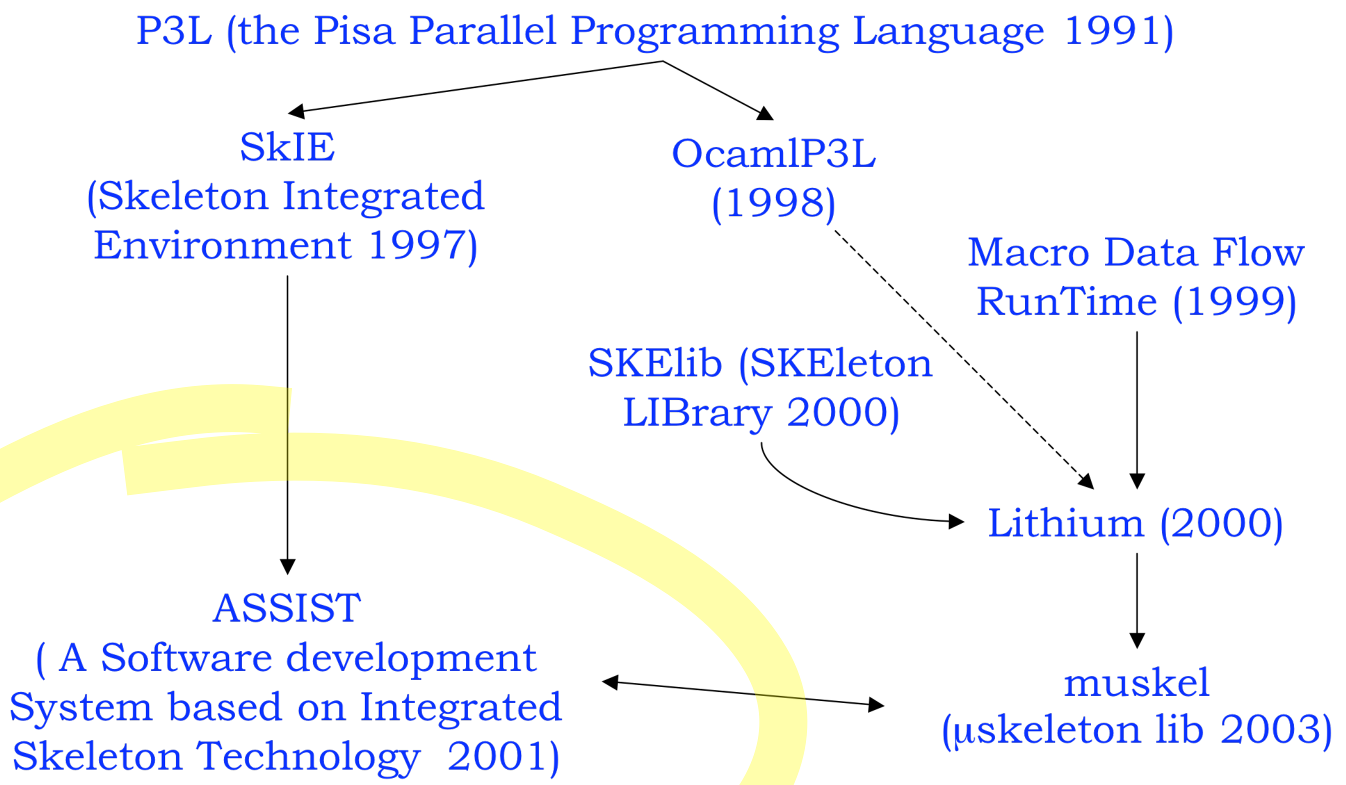
Fixed Skeleton Set vs. Parametric



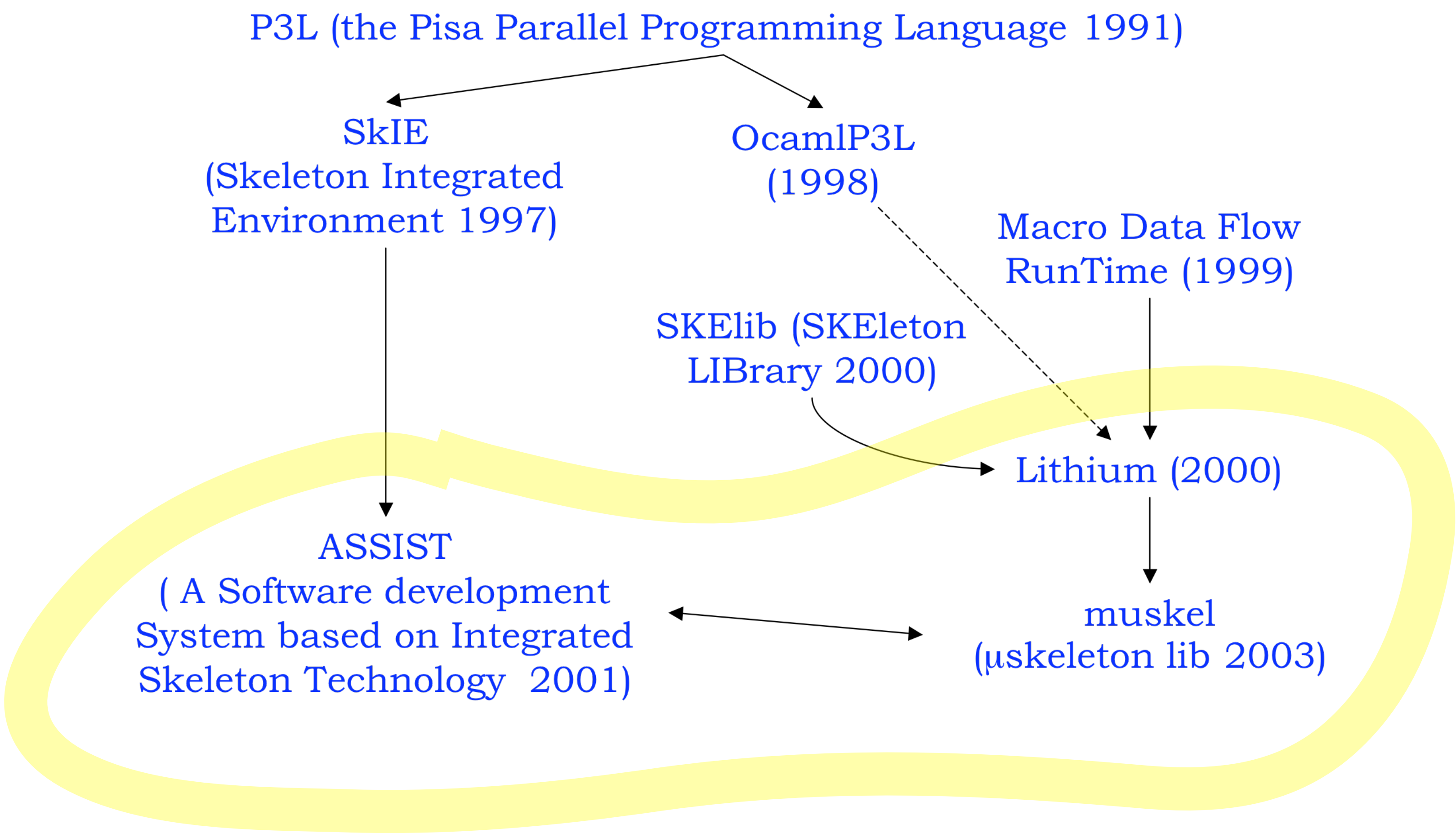
Code reuse vs. from scratch



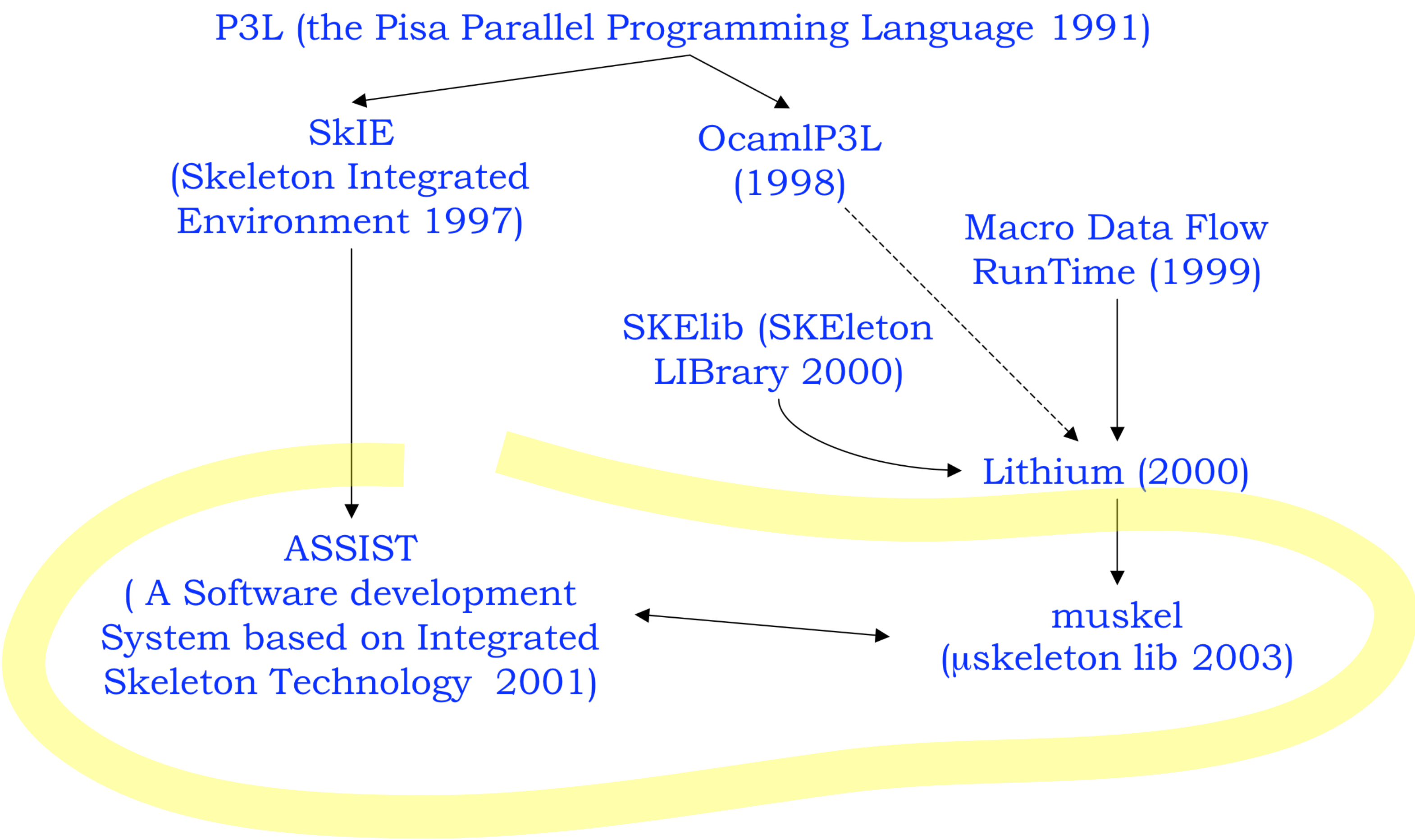
Interoperability



Heterogeneity



Dynamicity


P3L (the Pisa Parallel Programming Language 1991)
SkIE (Skeleton Integrated Environment 1997)
OcamlP3L (1998)
Macro Data Flow RunTime (1999)
SKElib (SKEleton LIBrary 2000)
Lithium (2000)
ASSIST (A Software development System based on Integrated Skeleton Technology 2001)
muskel (μskeleton lib 2003)

Lessons learned (before manifesto)

- Tools that allow code reuse have better agreement
- Tools that interact with "standard" environments have better agreement
- Programmers always want *just a little bit more* parallelism exploitation forms
- Libraries are easier to sell than languages
- Modern architectures *must* be targeted
  - Heterogeneous, dynamic, non exclusive usage, etc.

Cole's manifesto

❶ *Propagate the concept with minimal disruption*

- No chance to introduce yet another parallel programming language

❷ *Integrate ad hoc parallelism*

- Specialized, ad hoc solutions must be hosted

❸ *Accommodate diversity*

- Slightly different skeletons should be derivable

❹ *Show the payback*

- Advertising: demonstrate that moving to skeletons is worthwhile

Our additions

❺ *Support code reuse*

– Huge amount of (dusty deck?) code
– Large amounts of (open source) "libraries"

❻ *Handle heterogeneity*

– Cluster/networks/grids *are* heterogeneous
– Upgrades of clusters (with different release procs and different amounts/speed of main store)

❼ *Handle dynamicity*

– Non dedicated computing nodes (varying load)
– Different nodes, different power

Our claim

❶Minimal disruption
❷Ad hoc parallelism
❸Accommodate diversity
❹Show payback
❺Reuse code
❻Handle heterogeneity
❼Handle dynamicity

Skeleton systems (either skeleton based languages or skeleton libraries) can be classified as "*mature, second-generation*" if they satisfy the 7 requirements

Current skeletons in Pisa

- Learn from past experiences
- Address both Cole's manifesto and our additional requirements
- One experimental framework (muskel)
  - Compact, easy to modify, research only
  - Exploits Java to shortcut some problems (portability)
- One production framework (ASSIST)
  - Product of the GRID.it project
  - Huge, interoperable, runs on standard environments

muskel

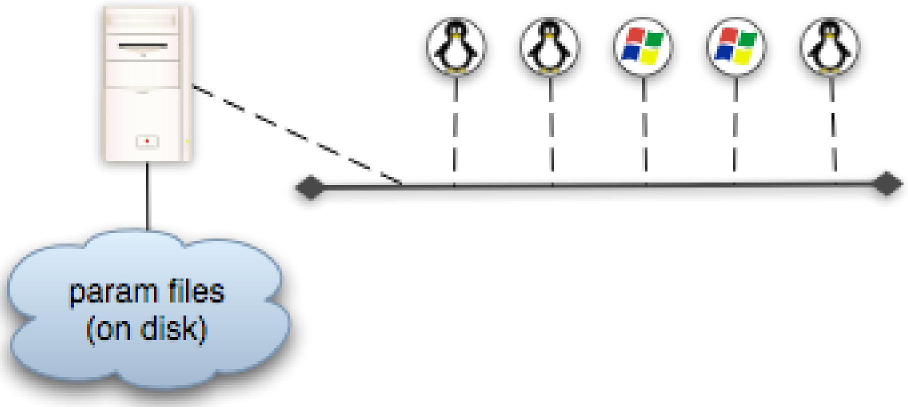
- μ-skeleton library → **muskel**
- Full Java library (❶ ❺)
- Small(est) subset of Lithium
  - Macro data flow (MDF) (❷ ❸)
  - RMI based remote execution
- P2P-like resource recruiting
  - UDP multicast (→ ?JXTA, → ?ProActive)
- With *application manager*
  - Ensuring user provided performance contract
  - Reacts to network / node faults
  - Looking for new resources
  - Rescheduling *lost* MDF instructions from scratch

❶Minimal disruption
❷Ad hoc parallelism
❸Accommodate diversity
❹Show payback
❺Reuse code
❻Handle heterogeneity
❼Handle dynamicity

muskel sample (1)

❶Minimal disruption
❷Ad hoc parallelism
❸Accommodate diversity
❹Show payback
❺Reuse code
❻Handle heterogeneity
❼Handle dynamicity

- Parameter sweeping application
- Parameter sets on disk files
- Set of remote WS accessible (❻ any OS with Java)
- *Target:* get a set of result files in the shorter time



`muskel` sample (2)

- On remote machines, run muskel run time (once and forall)
  - RemoteWorker java RMI server

- On the local machine just run (❶ ❺):

```
public static void main (String [] args)
{
  Compute mainProgram = new Farm(new doSweep());
  ParDegree parDegree =
        new ParDegree(Integer.parseInt(args[0]));
  ApplicationManager manager = new
  ApplicationManager(mainProgram);
  manager.setContract(parDegree);
  manager.inputStream(args[1]);
  manager.evalToFile(args[2]);
}
```

❶Minimal disruption
❷Ad hoc parallelism
❸Accommodate diversity
❹Show payback
❺Reuse code
❻Handle heterogeneity
❼Handle dynamicity

muskel sample (3)

```
Compute mainProgram = new Farm(new doSweep());
```
define program

```
ParDegree parDegree =
  new ParDegree(Integer.parseInt(args[2]));
```
prepare the perf contract to ask

```
ApplicationManager manager = new
  ApplicationManager(mainProgram);
```
instantiate a manger and tell it the program to be computed

```
manager.setContract(parDegree);
```
provide required performance contract

```
manager.inputStream(args[0]);
```
tell the manager where to find input tasks

```
manager.evalToFile(args[1]);
```
ask for program evaluation (res to file)

`muskel` implementation

- Basics
  - Centralized task pool hosts MDF instructions
  - One thread per remote worker:
    - Fetches a fireable instruction, invokes remote method to compute it, delivers resulting tokens to the proper place
  - Operates in <u>normal form</u> mode
    - Skeleton tree preprocessed to normal form, service time optimized
- Remote workers
  - Methods to accept MDF code (serialized)
  - Methods to compute MDF
  - Management methods (stats, load measure, etc.)

muskel implementation (2)

- Application manager (❼!!!)
  - Accepts performance contract
    - Parallelism degree (current)
    - Service time (forthcoming)
  - Discovers available resources
    - UDP multicast (current)
    - P2P (forthcoming)
  - Ensures fault tolerance
    - Faulty nodes replaced & tasks rescheduled
  - Ensures QoS
    - Nodes added if needed
    - Nodes released if steady over-contract

❶Minimal disruption
❷Ad hoc parallelism
❸Accommodate diversity
❹Show payback
❺Reuse code
❻Handle heterogeneity
❼Handle dynamicity

muskel & requirements

❶ Propagate the concept with minimal disruption	Plain Java library
❷ Integrate *ad hoc* parallelism	User access to streams & MDF level
❸ Accommodate diversity	User access to streams & MDF level
❹ Show the pay back	OO + rapid prototyping + efficiency
❺ Support code reuse	Java only
❻ Handle heterogeneity	By Java
❼ Handle dynamicity	Application manager

ASSIST

- Generic graph of either parallel or sequential modules
  - Sequential modules: C, C++, F77, (Java) (❺)
  - Parallel modules (❷ ❸):
    - Set of virtual processes (VP, named after a *topology*)
    - Program of the virtual processes as a seq module
    - (Possibly) sharing a state
    - Processing state and input data
    - To produce output data
    - SPMD is a sub case
- Interconnected via data flow streams
  - Non deterministic control over input streams
  - Input data scatter/uni-multi-broadcast to VPs

❶Minimal disruption
❷Ad hoc parallelism
❸Accommodate diversity
❹Show payback
❺Reuse code
❻Handle heterogeneity
❼Handle dynamicity

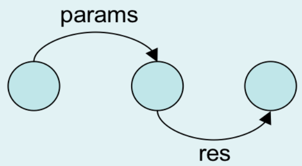
ASSIST (2)

❶Minimal disruption
❷Ad hoc parallelism
❸Accommodate diversity
❹Show payback
❺Reuse code
❻Handle heterogeneity
❼Handle dynamicity

- Interoperability (❺)
  - With WS & CORBA/CCM
  - To & From
    - ASSIST programs wrapper to CCM/WS (compile flag!)
    - CCM/WS accessible from within ASSIST programs (as modules in the graph or called from seq and VPs)
- GRID.it component model on top of ASSIST
  - Pipeline, farm, generic graph component
- Module (parmod) + application managers (❼)
- Runs on top of (❻)
  - Cluster/networks of TCP/IP POSIX workstations
  - Globus 2.4 grids

ASSIST sample code (1)

Define parallel module graph




```
generic main() {
    stream Param_t params;
    stream Res_t   res;
    generate_instream (output_stream params);
    doSweep(input_stream params output_stream res);
    process_outstream (input_stream res);
}
#pragma pardegree doSweep N
```

Set perf contract

```
parmod doSweep(input_stream Param_t param output_stream Res_t result) {
topology none my_vp;
do input_section {
 guard1: on true, MAX_PRI, param { distribution param on_demand to my_vp; }
} while (true)

  virtual_processes {
    computeSweep(in guard1 out result) {
      VP { doSweepSeq(in param out result); }
    }
  }
  output_section { collects result from ANY my_vp;}
}
```

Name processes (none = task farm)

Schedule input tasks

Define parallel module

Define logically parallel activities

Deliver computed results

ASSIST sample code (2)

```
proc generate_instream (output_stream Param_t params)
$c++{
  Param_t p;
  …
  p = … ;
  assist_out(params,p);
}c++$

proc process_outstream(input_stream Res_t res)
inc <iostream>
$c++{
  //  … some code processing res here …
}c++$

proc doSweepSeq(in Param_t p out Res_t r)
obj<myLibF-1.2.so>
$c{
  r = f(p);
}c$
```

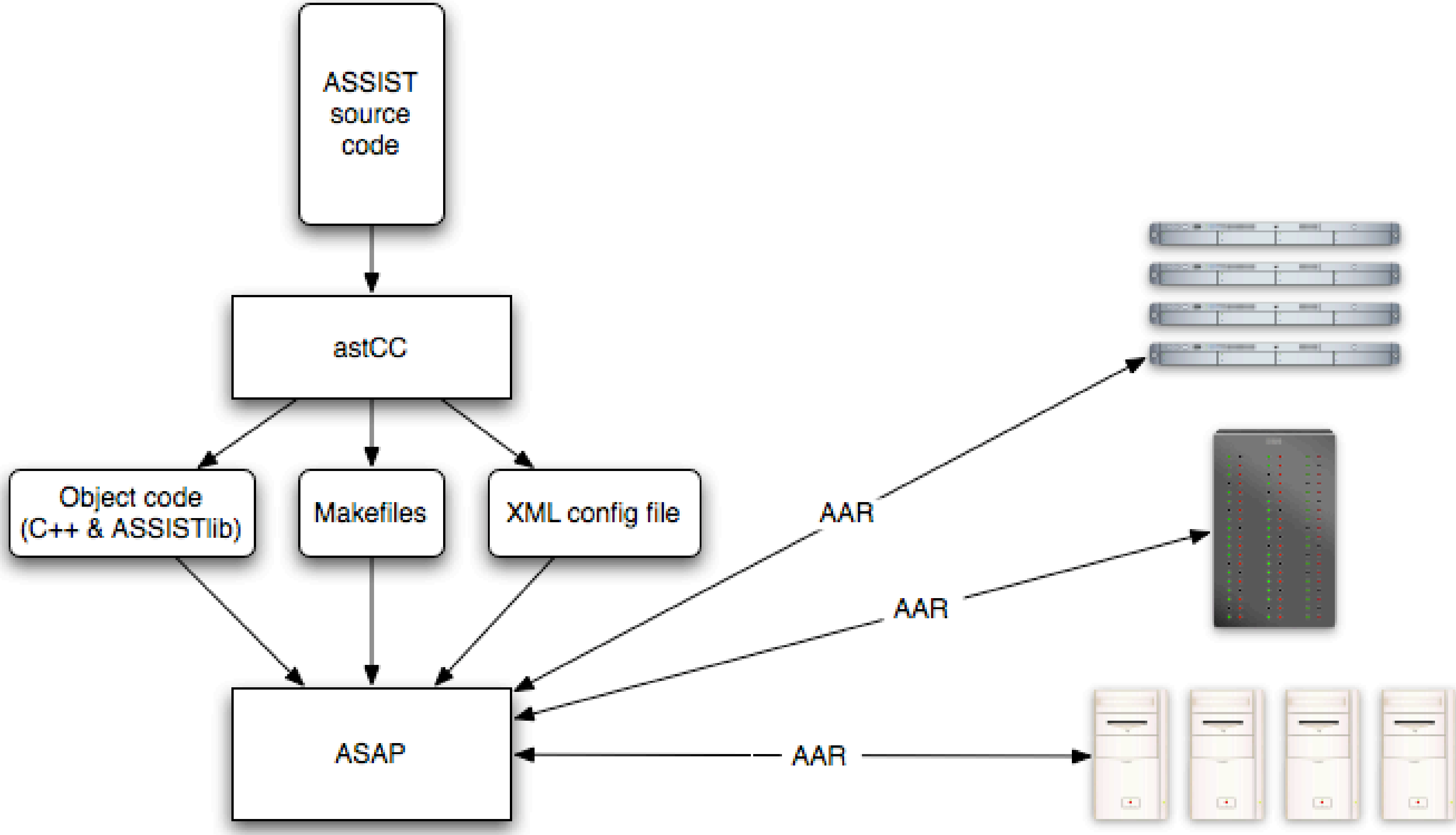
Specify seq language

Deliver items to output stream

Includes …

Link objects and libs …

ASSIST implementation



ASSIST & requirements

❶ Propagate the concept with minimal disruption	… definitely NO!
❷ Integrate *ad hoc* parallelism	Parametric parmod
❸ Accommodate diversity	Parametric parmod
❹ Show the pay back	Fairly fast application development + high efficiency
❺ Support code reuse	C C++ Fortran77
❻ Handle heterogeneity	Compiler + run time
❼ Handle dynamicity	Module & application manager

A final comparison

	muskel	ASSIST	eSkel	muesli
❶ Propagate the concept with minimal disruption	*Plain Java library*	…	*Plain MPI*	*Plain C++ & MPI library*
❷ Integrate *ad hoc* parallelism	*User access to streams & MDF level*	*Parametric parmod*	*Protected MPI communicators within skeletons*	*Variety of combinations of (data parallel) skeletons*
❸ Accommodate diversity	*User access to streams & MDF level*	*Parametric parmod*	*Parametric skeleton calls*	
❹ Show the pay back	*OO + rapid prototyping + efficiency*	*Fairly fast application development + high efficiency*	*Fast application development*	*OO library expressive power + fast development*
❺ Support code reuse	*Java only*	*C C++ Fortran77*	*C C++*	*C C++*
❻ Handle heterogeneity	*By Java*	*Compiler + run time*	*Guaranteed by MPI*	*Guaranteed by MPI*
❼ Handle dynamicity	*Application manager*	*Module & application manager*		

Conclusions

- Summary
  - Large experience with skeletons
- Requirements
  - Extending Cole's ones
- Current experiences in Pisa
  - Outlined
  - Related to requirements
  - And with other acknowledged skeleton systems

any questions ?

marcod@di.unipi.it

www.di.unipi.it/~marcod
(links to both muskel and ASSIST home pages)
(copy of these slides (soon))