PL efforts in UPMARC

an excerpt

Tobias Wrigstad

assistant professor

UPMARC

Uppsala Programming for Multicore Architectures Research Center

Short Bio

’06 PhD @ Royal Inst. of Technology, Sweden
’07 Postdoc @ Purdue University, IN US
’09 Associate prof @ Stockholm University
’10 Assistant prof @ Uppsala University

Aliasing in OOPLs
(Pluggable) type systems
Concurrent and parallel programming
Dynamic programming languages

```
080487dd: movl   $0x1,-0x20(%ebp)
080487e4: jmp    0x0804897c
080487e9: lea    0x0(%esi),%esi

080487ec: cmpl   $0x1,-0x1c(%ebp)
080487f0: jne    0x080488f0
080487f6: cmpl   $0x1,-0x20(%ebp)
080487fa: jne    0x080488f0
08048800: movzbl -0x422(%ebp),%eax
08048807: cmp    $0x6c,%eax
0804880a: je     0x080488ac
08048810: cmp    $0x6c,%eax
08048813: jg     0x08048838
08048815: cmp    $0x61,%eax
08048818: je     0x08048884
0804881a: cmp    $0x61,%eax
0804881d: jg     0x0804882c
0804881f: cmp    $0x22,%eax
08048822: je     0x0804885c
08048824: jmp    0x080488e8
08048829: lea    0x0(%esi),%esi
0804882c: cmp    $0x69,%eax
0804882f: je     0x08048898
08048831: jmp    0x080488e8
08048836: lea    (%esi),%esi
08048838: cmp    $0x6f,%eax
0804883b: je     0x080488d4
08048841: cmp    $0x6f,%eax
08048844: jg     0x08048850
08048846: cmp    $0x6d,%eax
08048849: je     0x08048870
0804884b: jmp    0x080488e8
08048850: cmp    $0x74,%eax
08048853: je     0x080488c0
08048855: jmp    0x080488e8
0804885a: lea    (%esi),%esi
0804885c: cmpb   $0x6d,-0x421(%ebp)
08048863: je     0x0804886c
08048865: movl   $0x0,-0x20(%ebp)
0804886c: jmp    0x080488e8
0804886e: lea    (%esi),%esi
08048870: cmpb   $0x61,-0x421(%ebp)
08048877: je     0x08048880
08048879: movl   $0x0,-0x20(%ebp)
08048880: jmp    0x080488e8
08048882: lea    (%esi),%esi
08048884: cmpb   $0x69,-0x421(%ebp)
0804888b: je     0x08048894
0804888d: movl   $0x0,-0x20(%ebp)
08048894: jmp    0x080488e8
08048896: lea    (%esi),%esi
08048898: cmpb   $0x6c,-0x421(%ebp)
0804889f: je     0x080488a8
080488a1: movl   $0x0,-0x20(%ebp)
080488a8: jmp    0x080488e8
```

Spaghetti Code
(~1968)

Spaghetti Data
(present day)

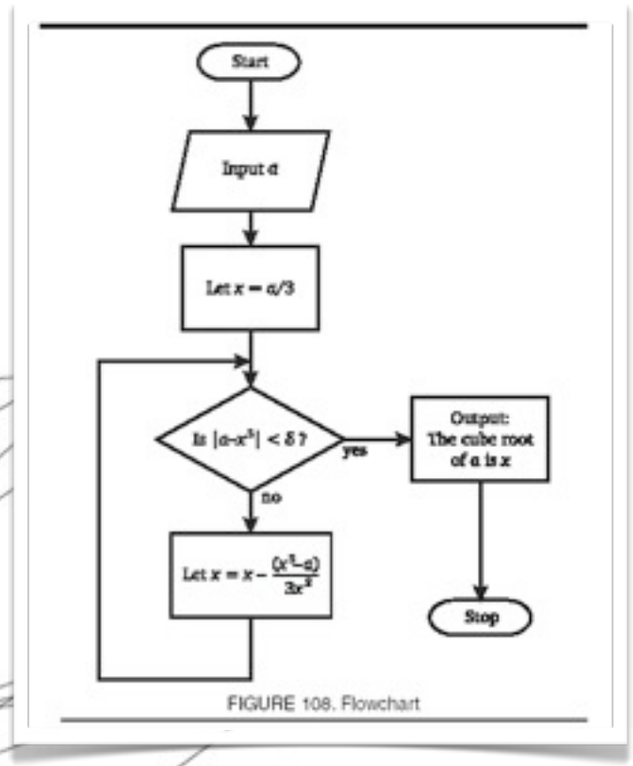
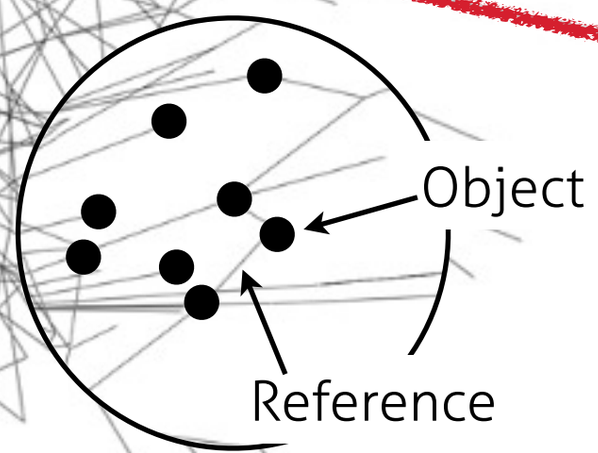


FIGURE 108. Flowchart

Races etc.

Thread A

Thread B



Visualisation of a heap from an OO program

Capsule Summary

Spaghetti data + ubiquitous parallelism is a no-go

— However, sharing state is key to performance!

Our goal

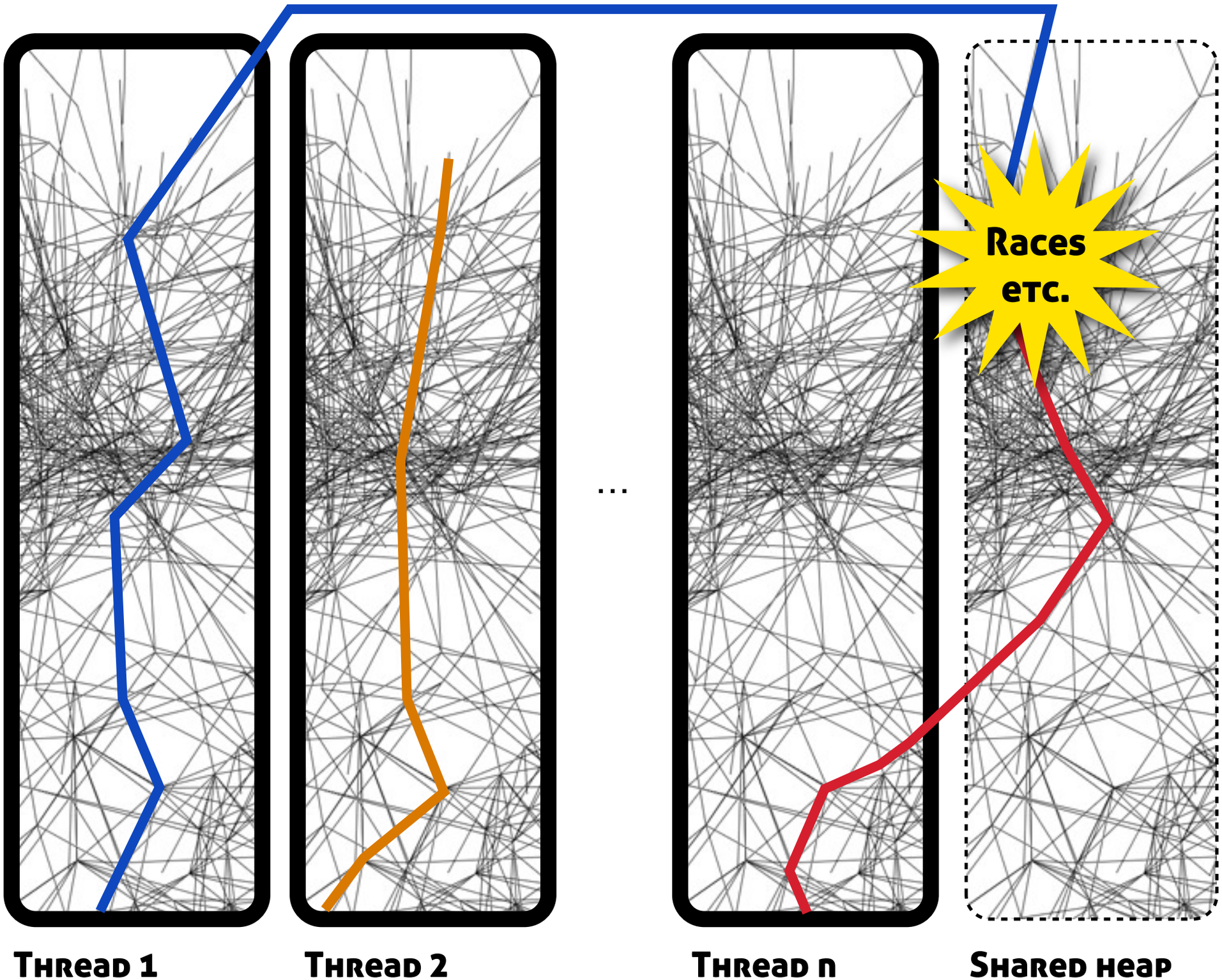
- Bring order to object-oriented data

Our approach

- Enable programmers to express sharing, locality, etc.
- Use this information for checking and parallelisation
  (correctness) *(efficency)*
- Avoid problems (races, compositionality, etc.) by design

Thread-Local Heaps in Loci [ECOOP 2009]

1ST PROPOSAL



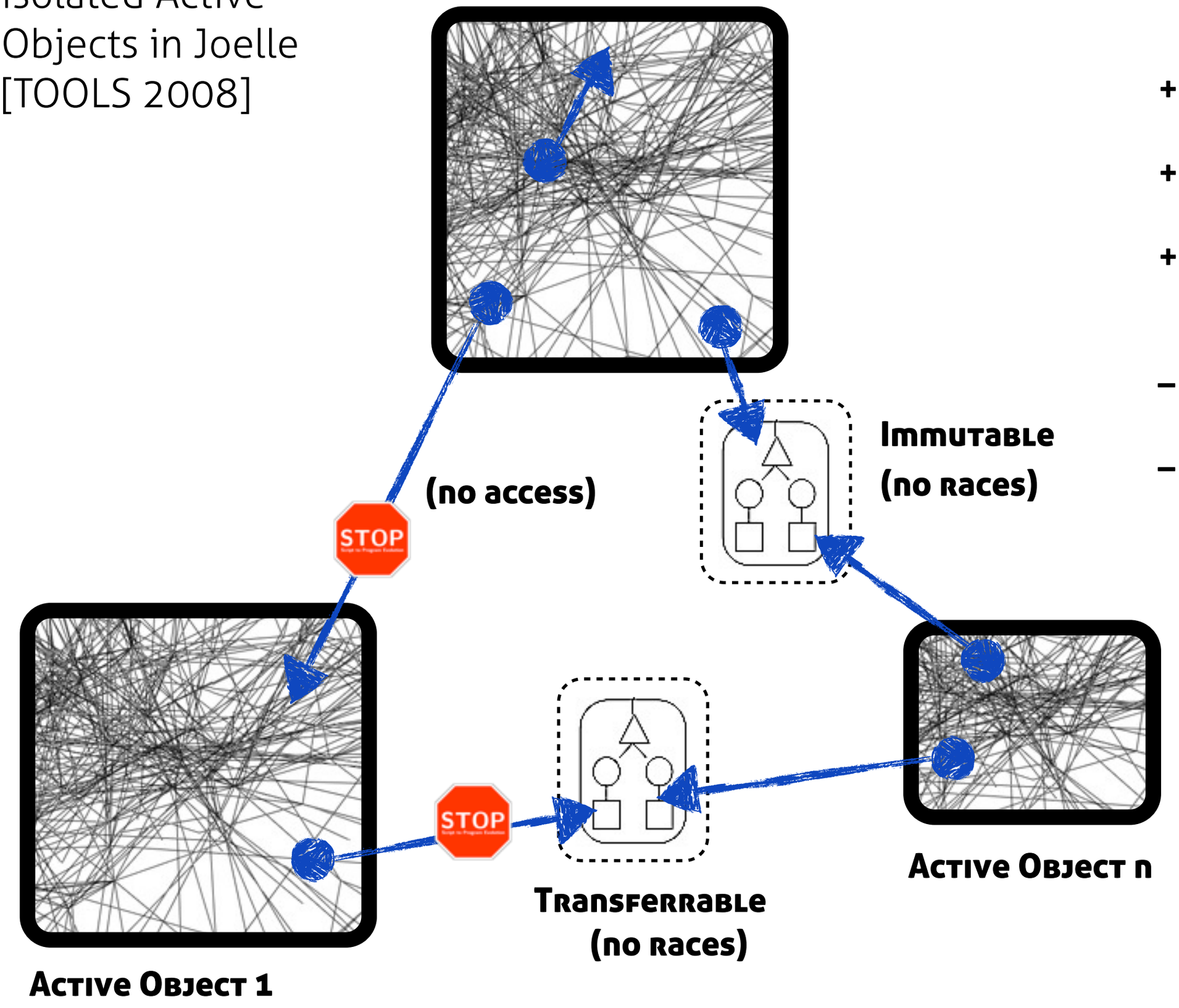
+ Shared accesses identified
+ Fine grained
+ Pluggable
+ Low syntactic overhead

– Shared heap still a mess
– Locks are not compositional

http://loci.java.net/

2nd Proposal

Isolated Active Objects in Joelle [TOOLS 2008]



+ Compositional

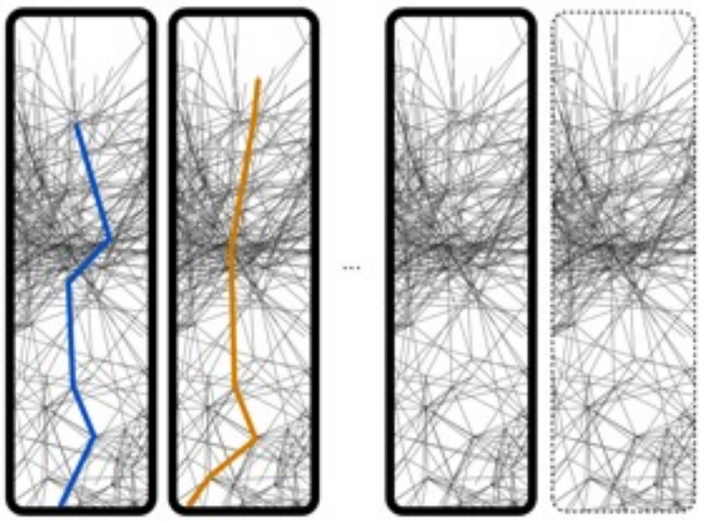
+ No races

+ Low syntactic overhead

– Coarse grained

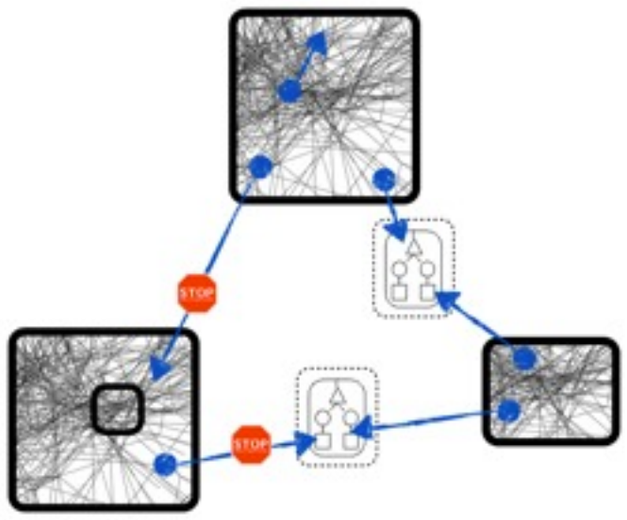
– No internal parallelism

The Way Forward: Our Core Design



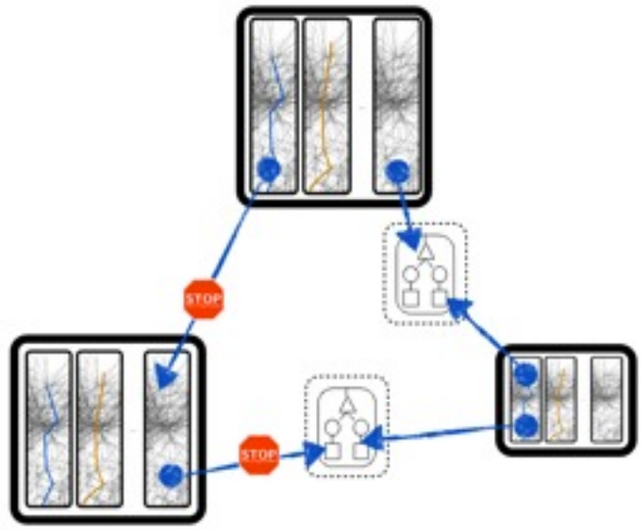
Loci

+



Joelle

=



Joelle 2.0

Ultimate Goals

- Replace Java as the safe mainstream programming language

  Simple & gradual system, legacy, do not ignore the programming craft...

- Deal with parallelism and concurrency better than Erlang

  Efficiency, locality, migration, high-level errors only...

- Without compromising with object-orientation

  Support shared mutable state as effortlessly as possible, encapsulation...

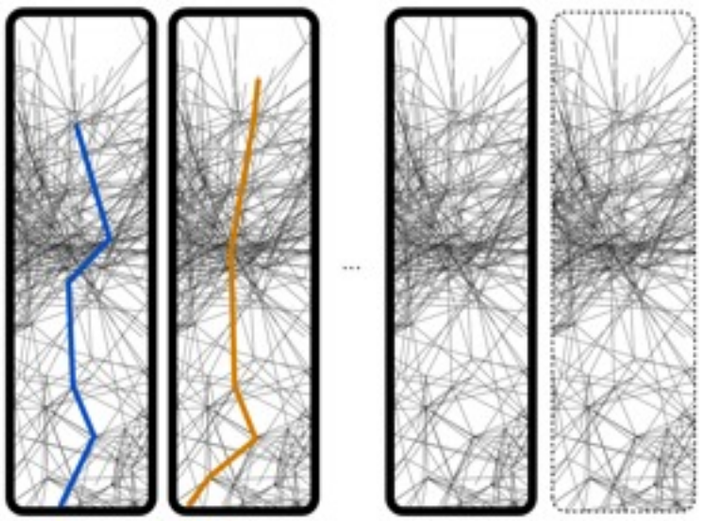
Current Goals

- Avoid races, deadlocks, subtle memory model considerations
- Better utilise shared resources (caches, shared buffers, etc.)
- Implicit parallelism where possible
- Facilitate manual and automated reasoning
- Language a superset of Java (or C++)

Interested industry: Ericsson, ABB, IBM, Oracle (no real committment from anyone yet)

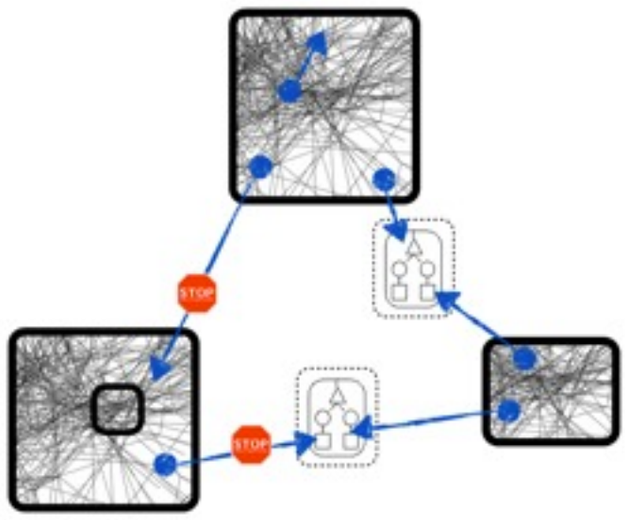
Impact: OOPSLA 2010, ECOOP 2009, APLAS 2008, TOOLS 2008
(Community: **sc** IWACO '07–11; **is** UPMARC summer school '10; **pc** FTfJP '11, IWACO '11, OOPSLA '11, ECOOP '12 **je** LNCS state-of-the-art aliasing in OOP Journal)

The Way Forward: Our Core Design



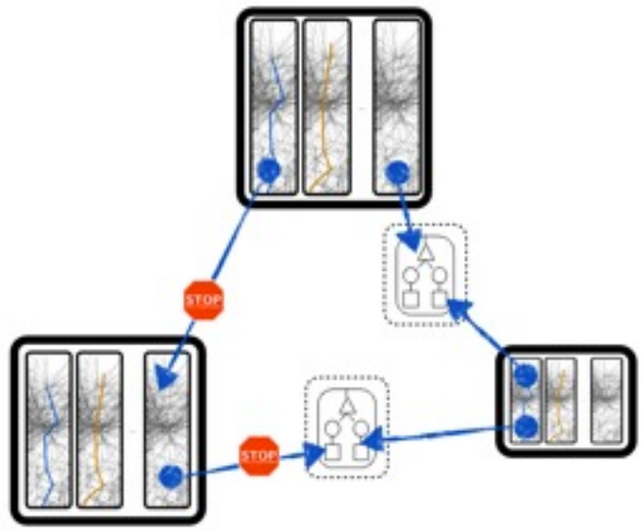
Loci

+



Joelle

=

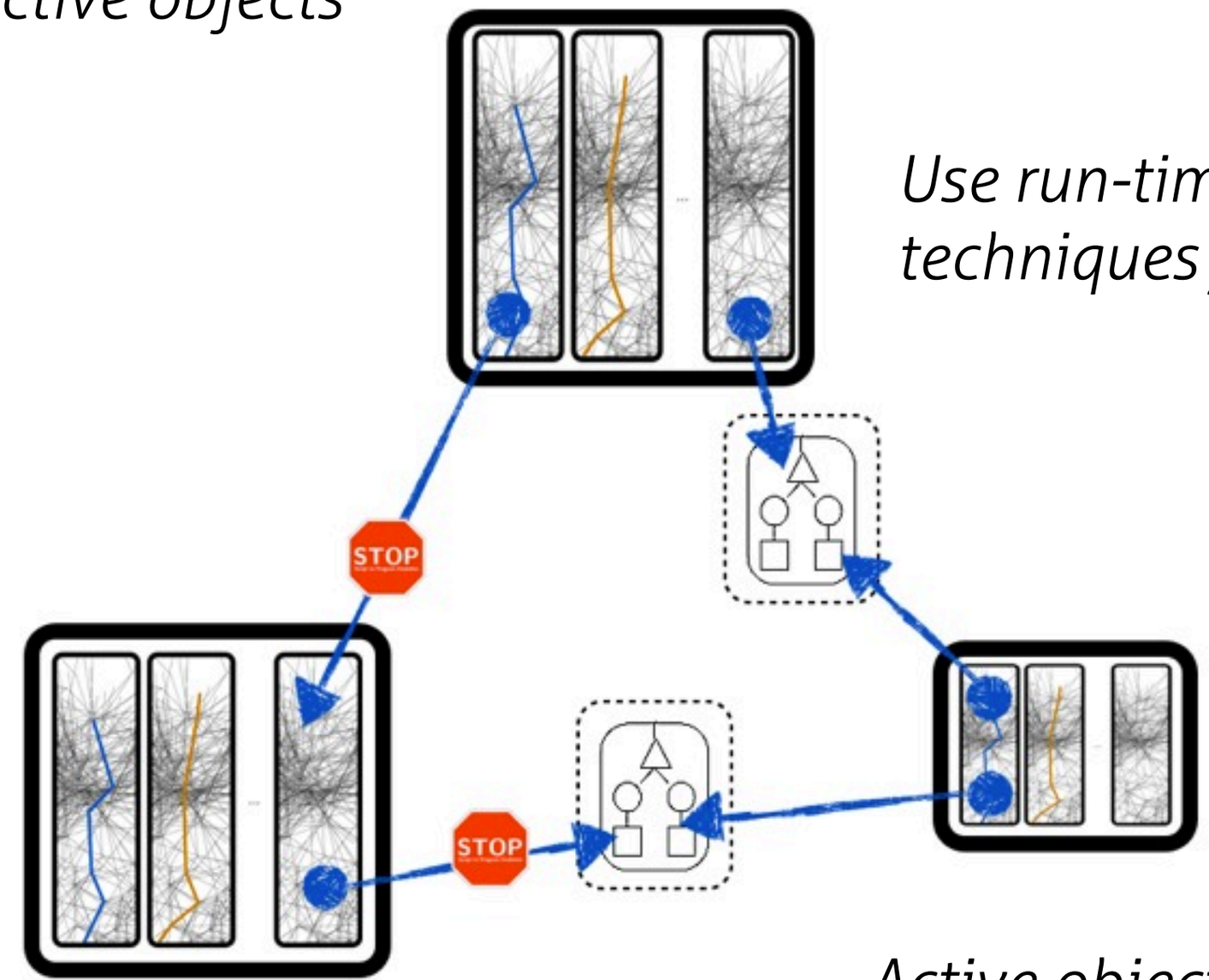


Joelle 2.0

Ownership types *Effect systems* *Alias Analysis* *Inference*

New!

Fine-grained parallelism inside active objects



Use run-time adaptation techniques for performance

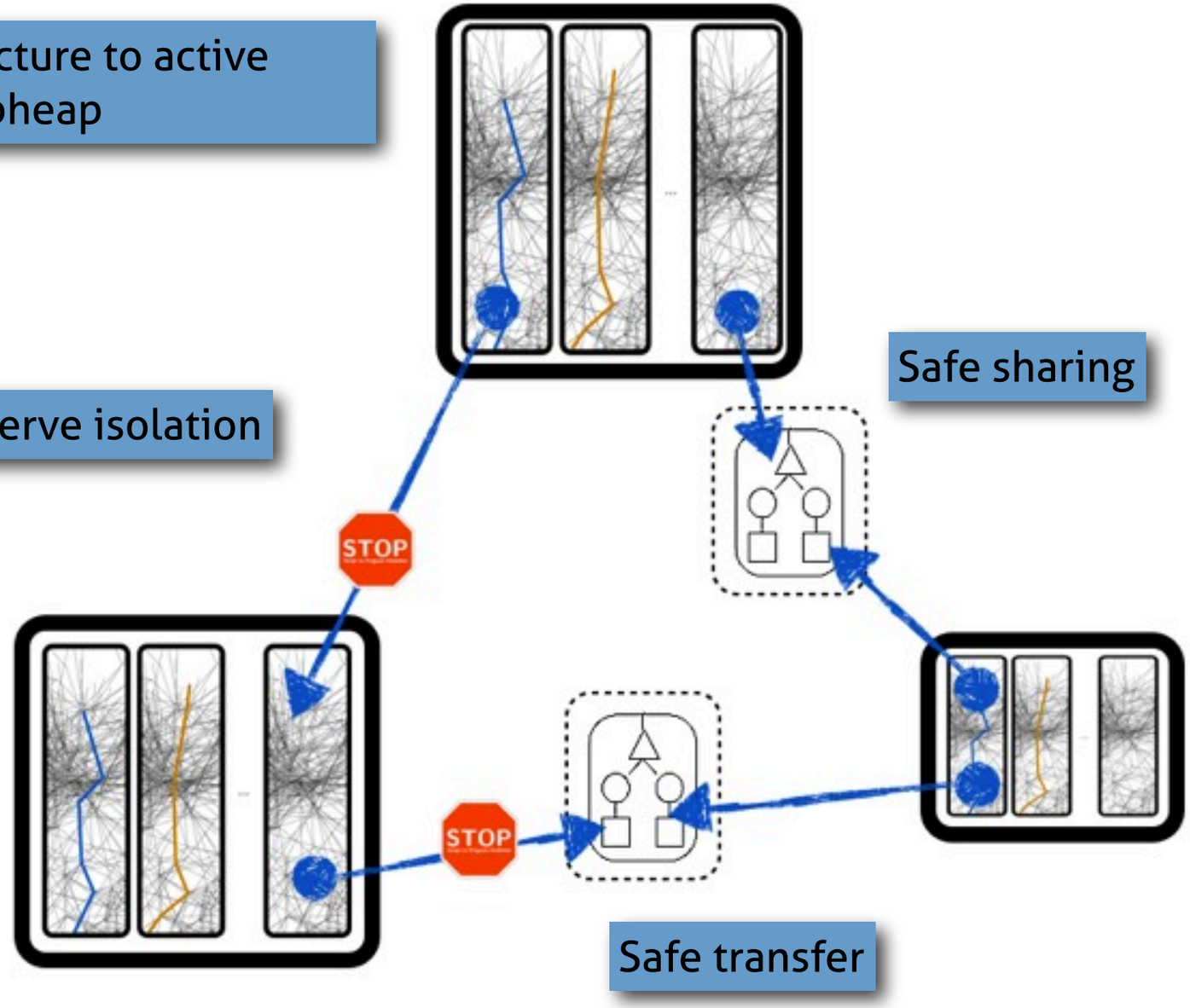
Active object model is safe by design

New!

Static

Dynamic

Bring structure to active object subheap



Preserve isolation

Safe sharing

Minimal clone operations

Safe transfer

Ownership types *Effect systems* *Alias Analysis* *Inference*

New!

Static

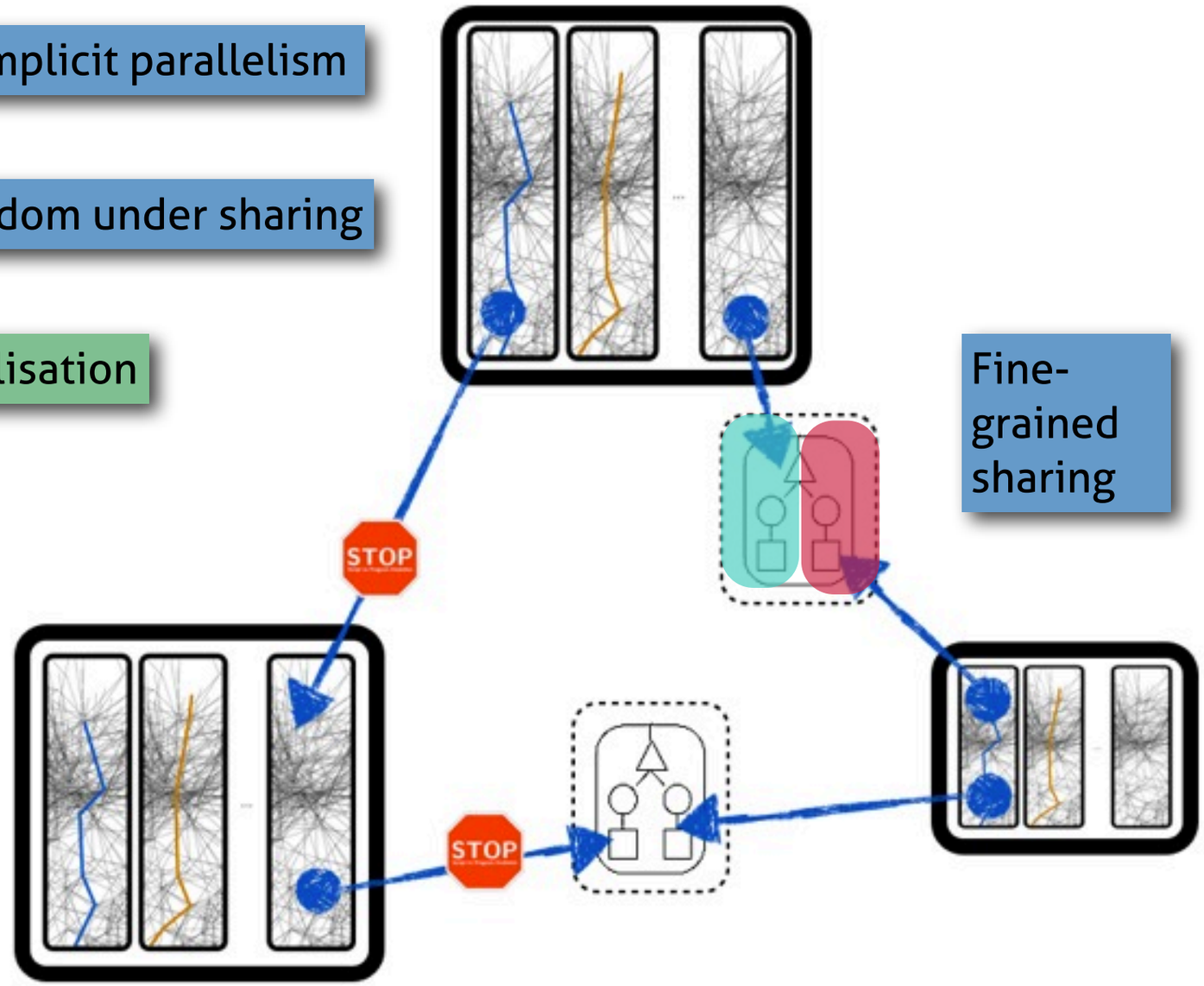
Dynamic

Allow implicit parallelism

Guarantee race freedom under sharing

Scheduling to maximise cache utilisation

Prefetching from message queue analysis



Fine-grained sharing

Ownership types **Effect systems** *Alias Analysis* *Inference*

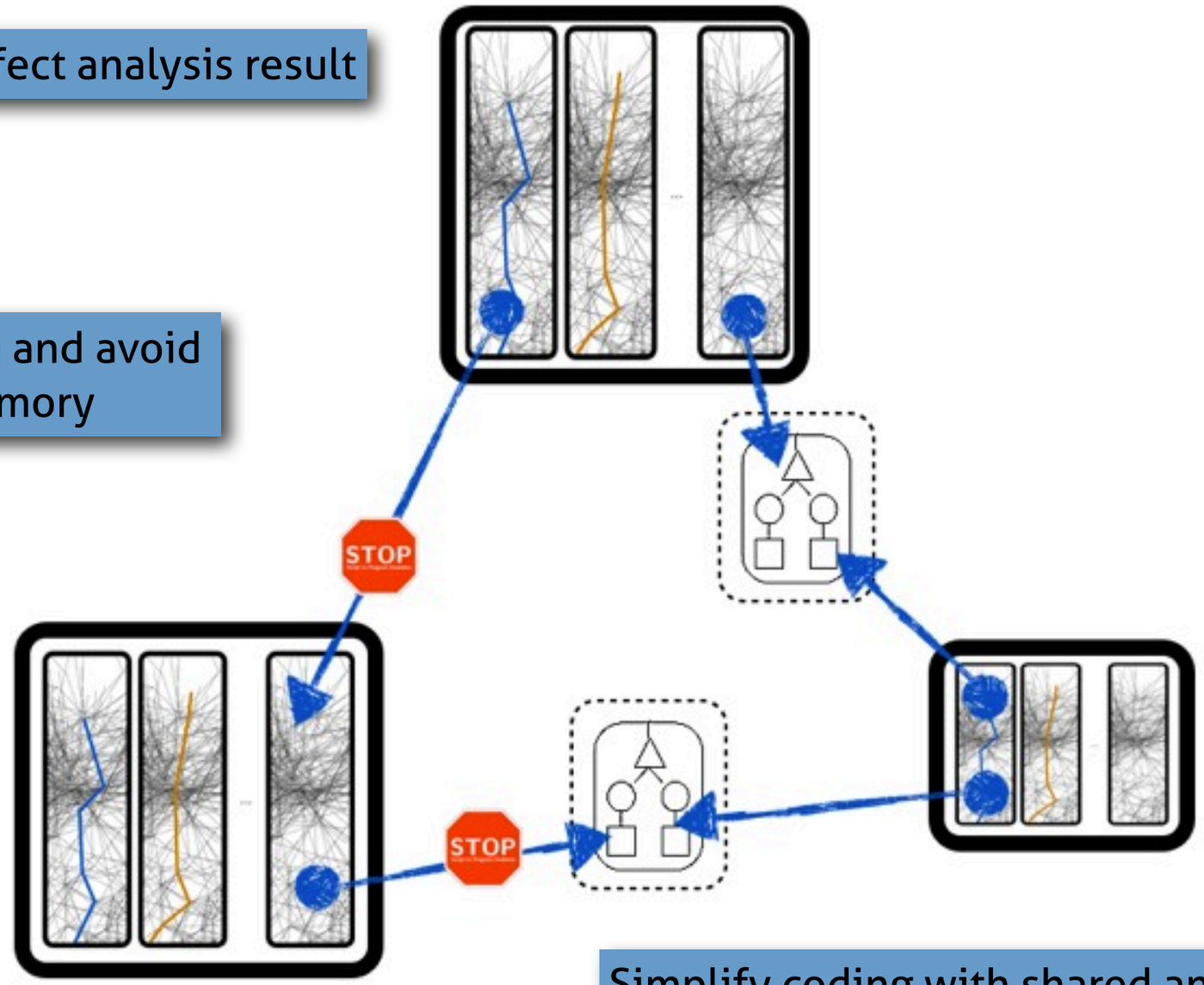
New!

Static

Dynamic

Improve effect analysis result

Compile fast-path and avoid synch to main memory



Simplify coding with shared and unshared values

Ownership types *Effect systems* **Alias Analysis** *Inference*

Static

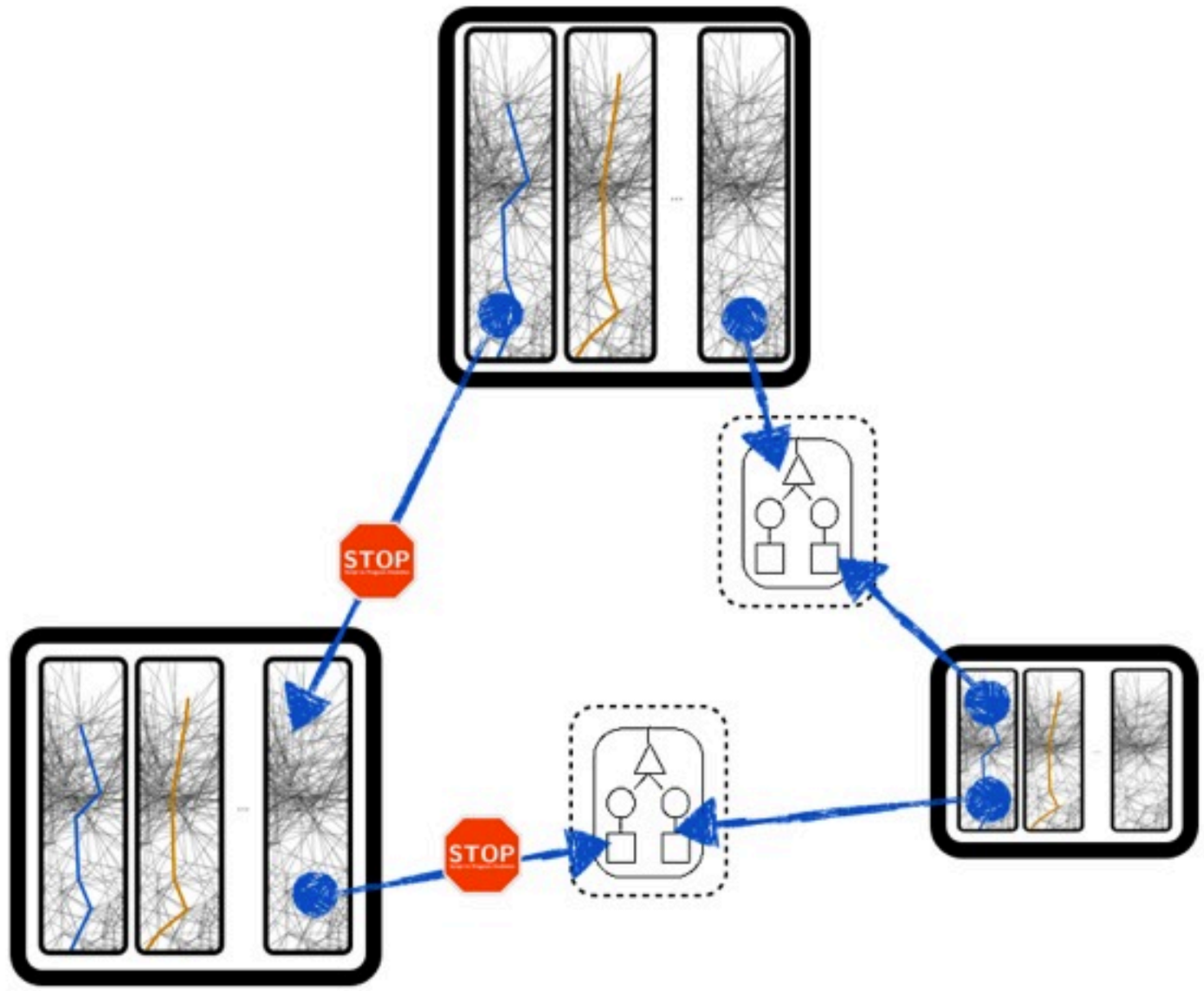
Dynamic

Lighten annotation burden

Facilitate reuse

Facilitate refactoring

New!

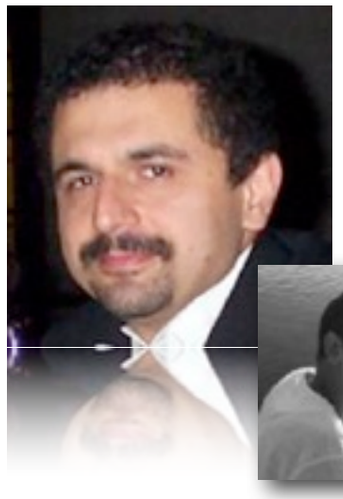


Ownership types *Effect systems* *Alias Analysis* **Inference**

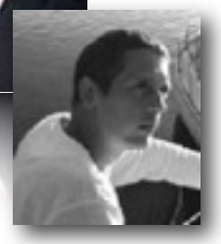
Immediate UPMARC Synergy

Programs in our system exhibit strong properties that facilitate program analysis

- e.g., alias freedom, locality information, effects of expressions
- Previous work on verification @ UU could capitalise on this



Parosh Abdulla



Jonathan Cederberg

Migrating legacy code to active objects

- Can it be done automatically?
- Inference in isolated enclosures — a smaller problem?
- Maybe annotations can be partially inferred?

Ownership and effect information used for scheduling, resource management

Wang Yi

David Black-Schaffer

Obstacles & Some Open Questions

Need more “warm bodies” (PhD students & PostDocs)

Need representative legacy code (tentative from ABB)

What are the effects on common idioms and programming practises?

Inference vs. programmer annotations—what is a good balance?

- Inference is flexible but brittle
- Annotations are stable but stale

Will a single active object concept fit all circumstances?

How to feedback that we cannot run something in parallel to the programmer?

…

Related Work (Excerpt)

Proactive, Scoop, Akka, etc. — active object systems for Java, Eiffel, Scala; no isolation guarantees, not for parallel programs

DPJ — share some ideas but for threads and task-based parallelism only; extreme programming overhead

X10 — captures where a computation takes place in the "place type"

CoBoxes, JCoBoxes — similar ideas for encapsulation but completely dynamic

FlexoTask and StreamFlex — shares ideas for encapsulation but for stream programming

A wealth of systems for ownership types, linearity, effect systems (Clarke, Boyland, Noble, Vitek, Aldrich, Rinard, Liskov, ...)

Ownership types inference work by Milanova et al., Ma & Foster

Jade — implicit parallelism; Futures; Erlang; ...

Our Unique Footprint:

Flexible yet powerful aliasing constraints; ownership-based effects; 2D rep splitting

Active-object based

Combination of annotations and inference; full static checking

Both coarse-grain (AO) and fine-grain (task, etc.) parallelism

Run-time reliance on ownership and effects for scheduling and implicit parallelism

Consider programming surroundings & legacy

Thank you! Questions?

UPMARC Uppsala Programming for Multicore Architectures Research Center