A String of Ponies

Transparent Distributed Programming with Actors

Sebastian Blessing

Supervised by Prof. Sophia Drossopoulou and Sylvan Clebsch

M.Sc in Computing Science (Software Engineering), 13. September 2013

(Concurrent) Pony

- Object-oriented
- Actor-based with *causal messaging*
  - Zero-copy message passing semantics
- Fully concurrent garbage collection of actors and objects
  - No “stop-the-world“ step necessary
- One kernel thread per core
- Work stealing scheduler
- Termination based on *quiescence*

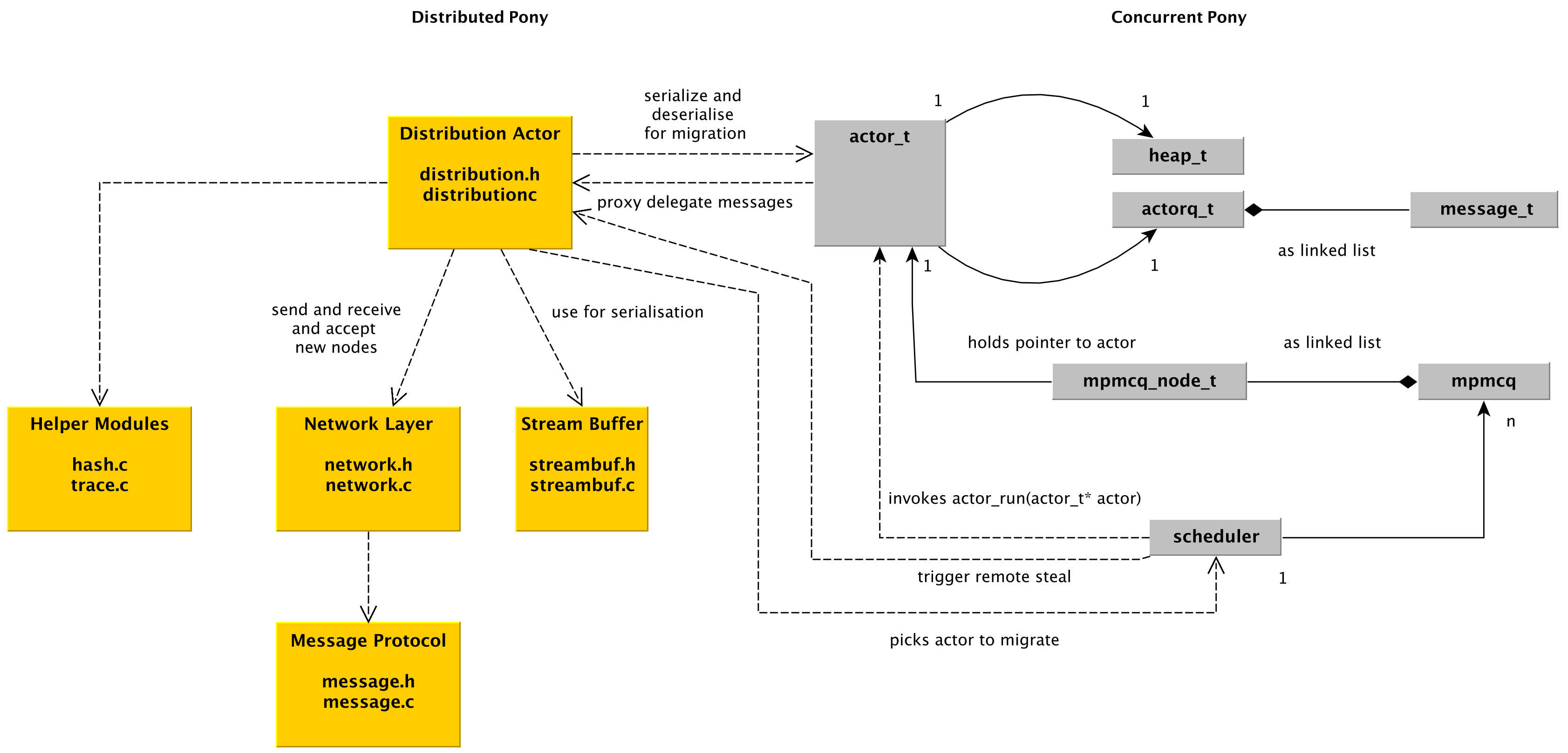
Extend Pony with *transparent* Distribution

- Distribution through Actor *mobility*

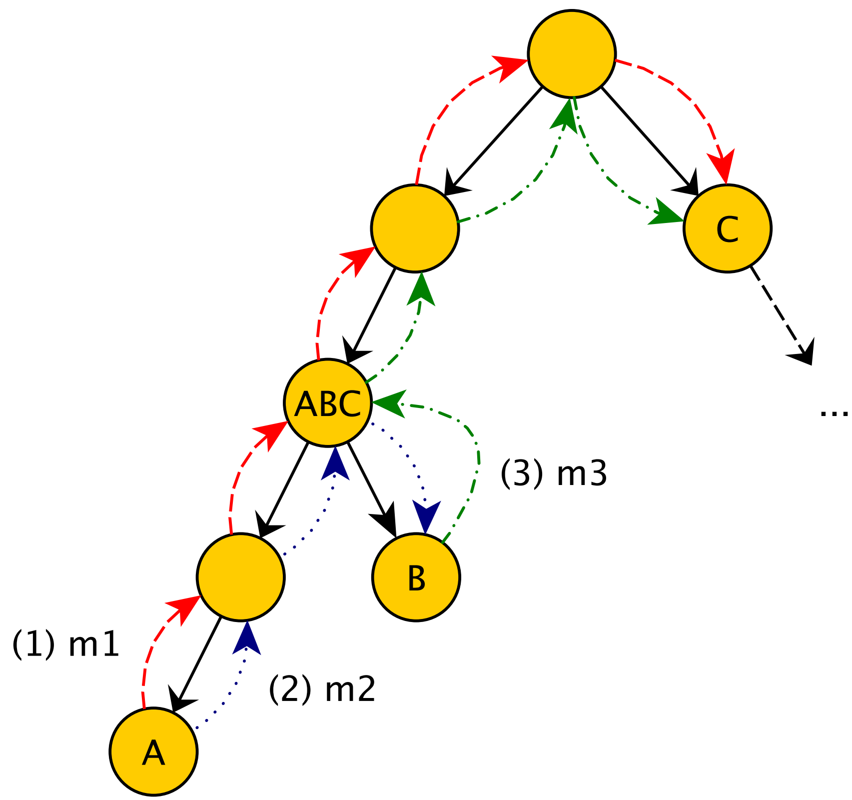
- Requirements and Contributions
  - Networking Layer (Async. I/O multiplexing)

1. Causal Messaging in Distributed Systems
2. Join nodes to a cluster of *Ponies*
3. Distributed Work Stealing Scheduler
   a. Proxies and causality
4. Serialisation and Deserialisation
5. Distributed Garbage Collection of Actors and Objects
6. Distributed Termination

Architectural Overview



1. Causal Messaging in Distributed Systems



Causality: m1 -> m3 at C

3. Distributed Work Stealing

- **Challenges**:
  - Detect resources that or not used to capacity
  - Migrate Actors to nodes in a network

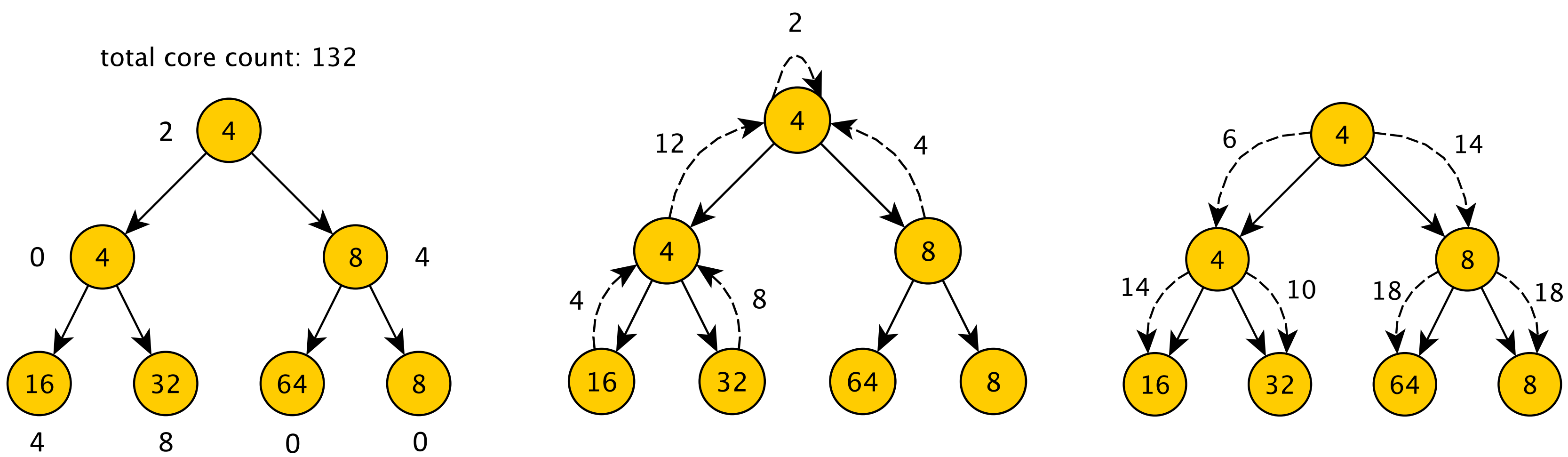
- **Problem:** busy nodes should not cause their children to suffer from starvation

- Leverage tree topology

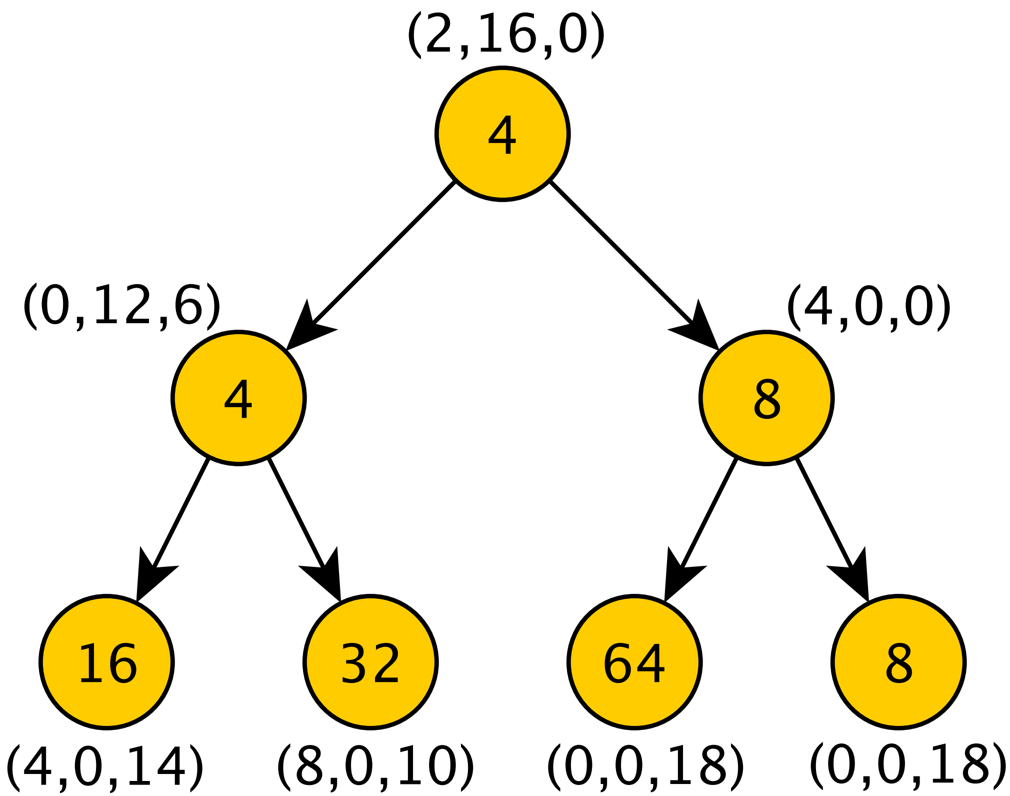
3. Distributed Work Stealing

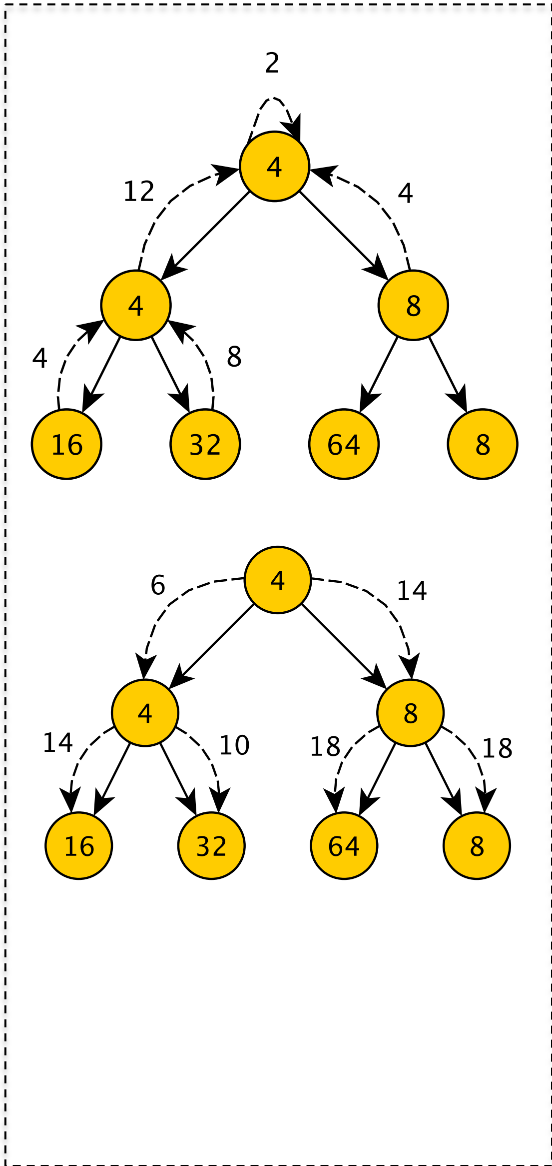
Dashed UP arrows: report free core counts (transitively)

Dashed DOWN arrows: report free cores reachable via parent

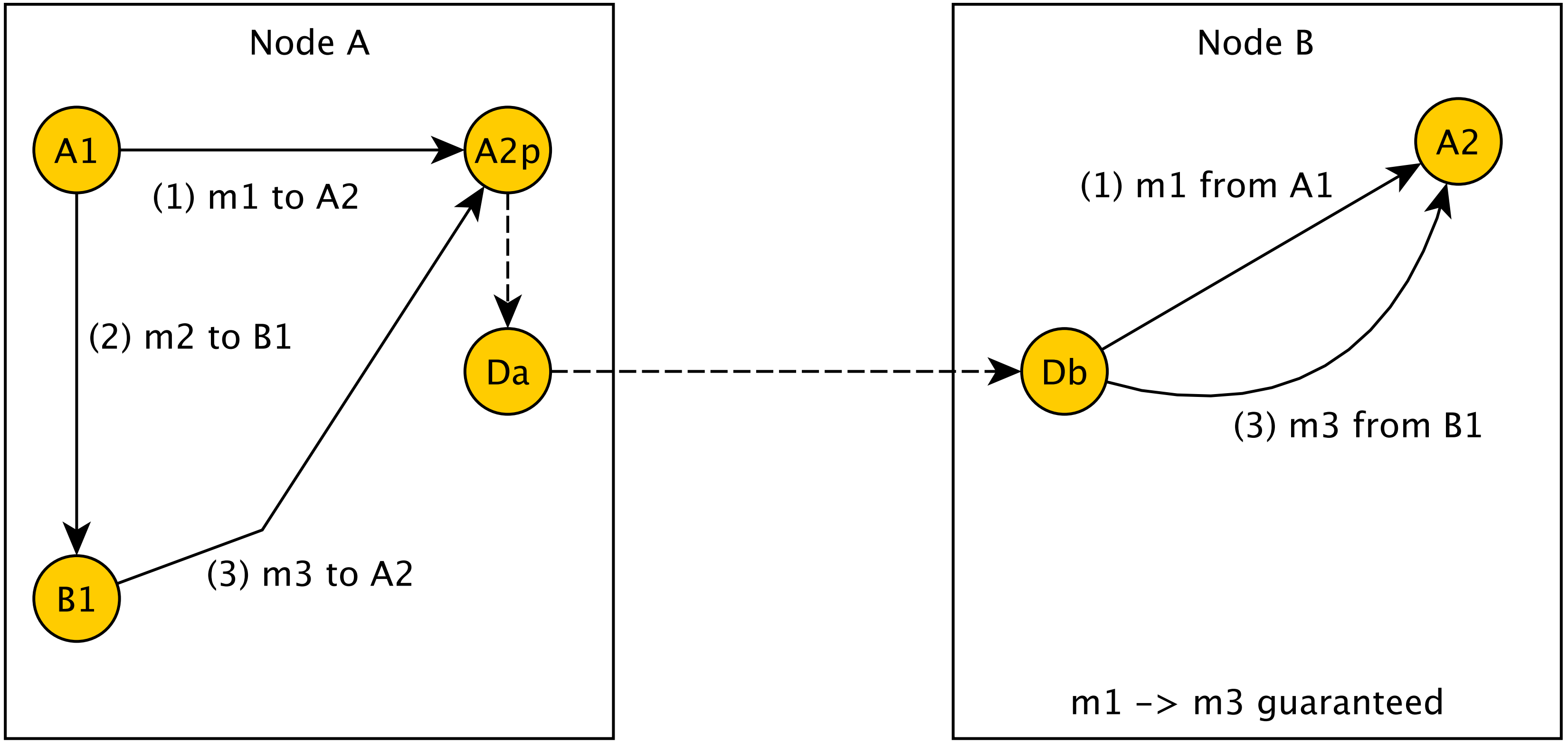


3. Distributed Work Stealing

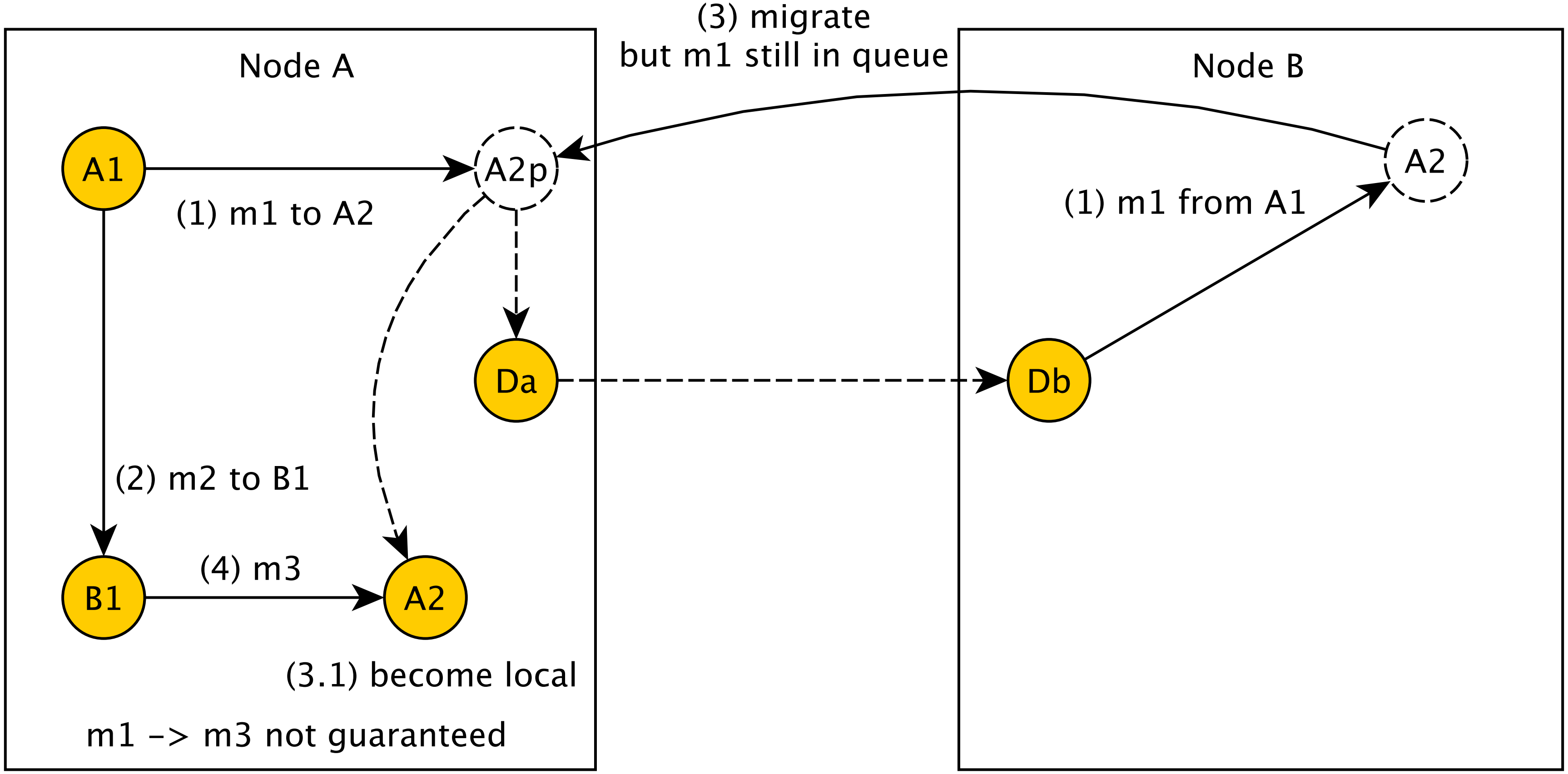




3a. Migrated Actors become *Proxies*



3a. Problem: Migration can *break causality*



3a. Answer: Causality-aware Migration Protocol

- Force A2 to have processed m1 *before* migration
- Introduce *conf-ack* cycle
  - Stop delegating messages once proxy received conf. message
  - Migrate when remote actor receives ack. message

5. Garbage Collection: Cycle Detection

- (Concurrent) Pony employs a *cycle detector* (implemented as actor)

- *A blocked* actor sends its ID, its ref-count and the set of actors it references to the cycle detector
- If an actor unblocks, it sends UNBLOCK(ID) to the cycle detector
- Cycle detector determines *perceived* actor state

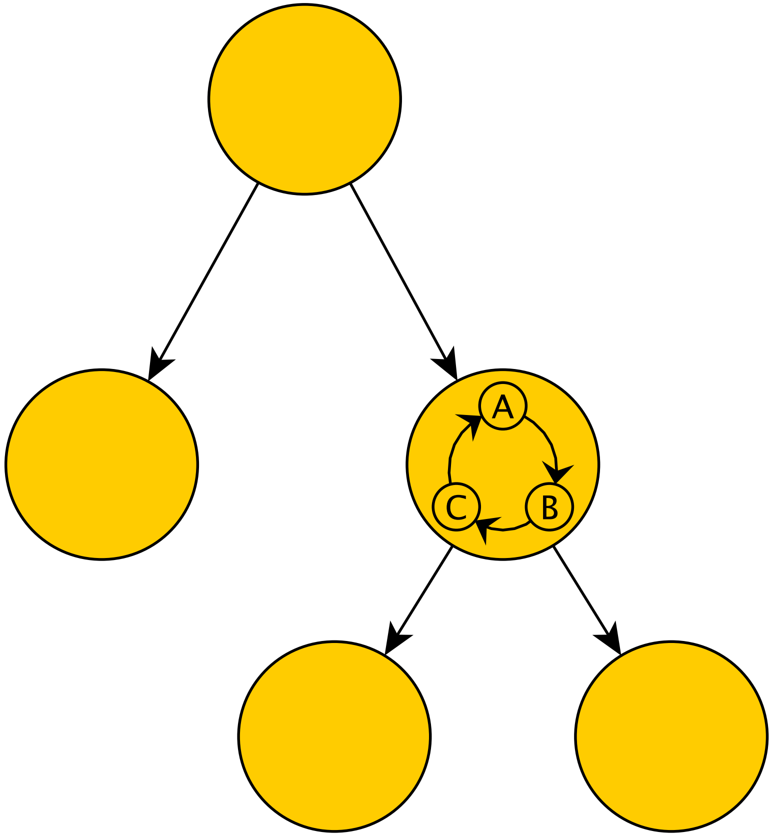
- Problem: *Asynchronicity*, actor states may be out of sync.

5. True Cycle Detection Protocol

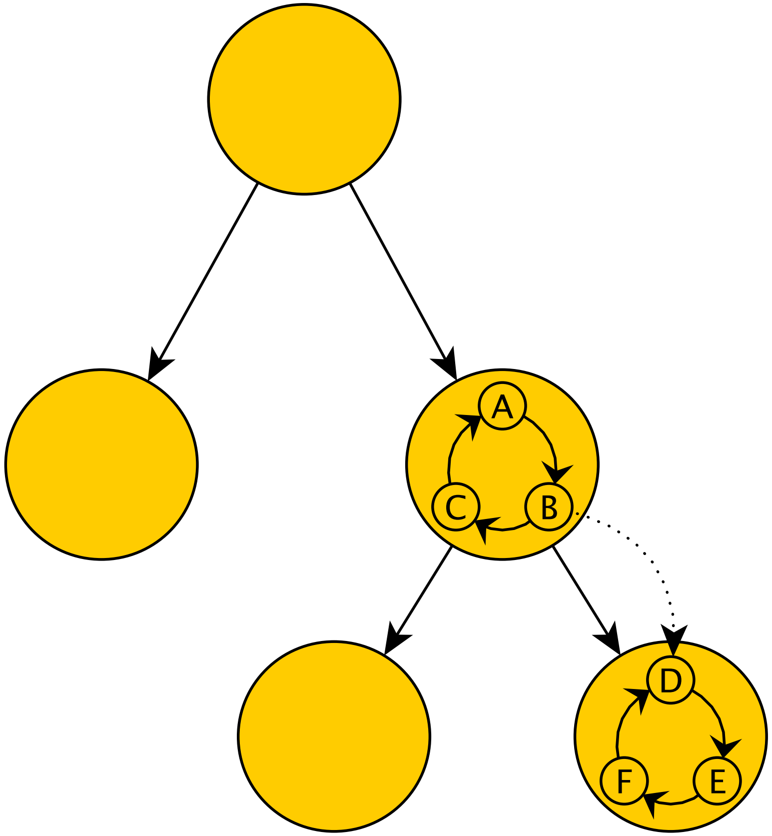
- Cycle detector sends CONF(t) to all actors in a perceived cycle
- Actors respond with ACK(t)

- *Causality* helps:
  - If the cycle detector received ACK(t) from all actors in *t* without having received UNBLOCK(a) after CONF(t) from some actor, the perceived cycle must be a true cycle and can be collected

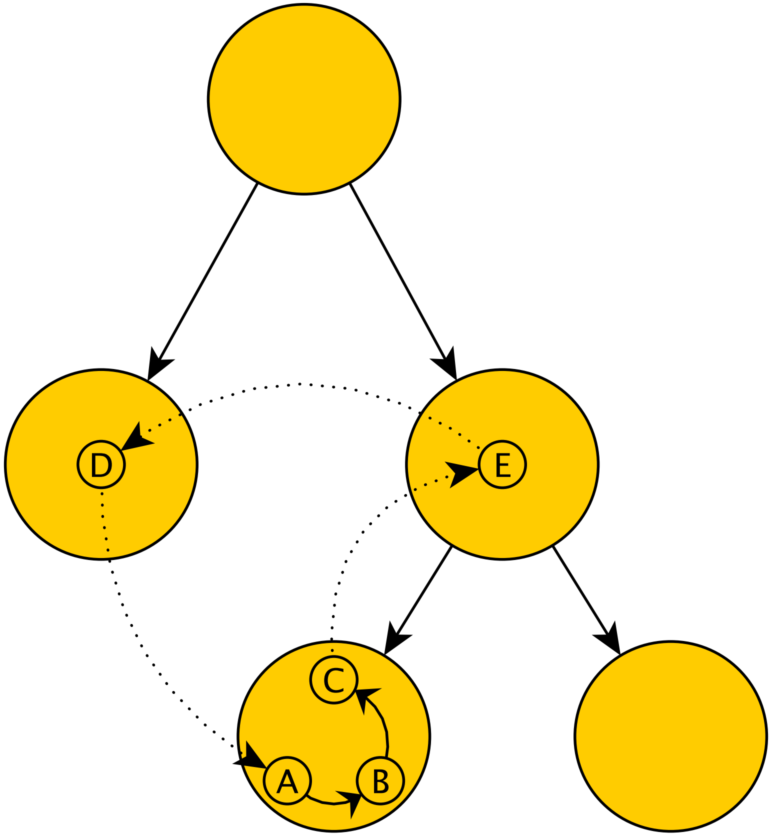
5. Distributed Cycle Detection



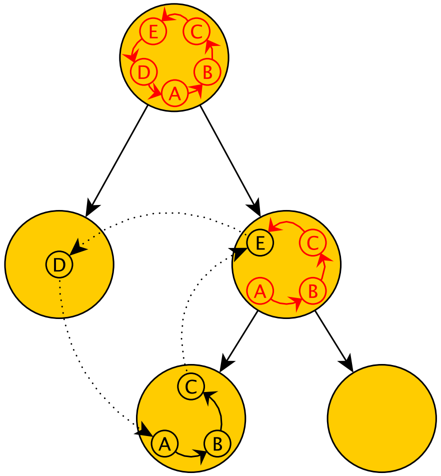
5. Distributed Cycle Detection



5. Distributed Cycle Detection



5. Distributed Cycle Detection



6. Distributed Termination

- Re-use conf-ack protocol from cycle detection
- Employ central *termination detector* on the master node

- Nodes starved from work report “idle“
- Once work is available report “busy”

- The termination detector having received “idle” from all nodes (including the master) sends CONF(t) to all nodes

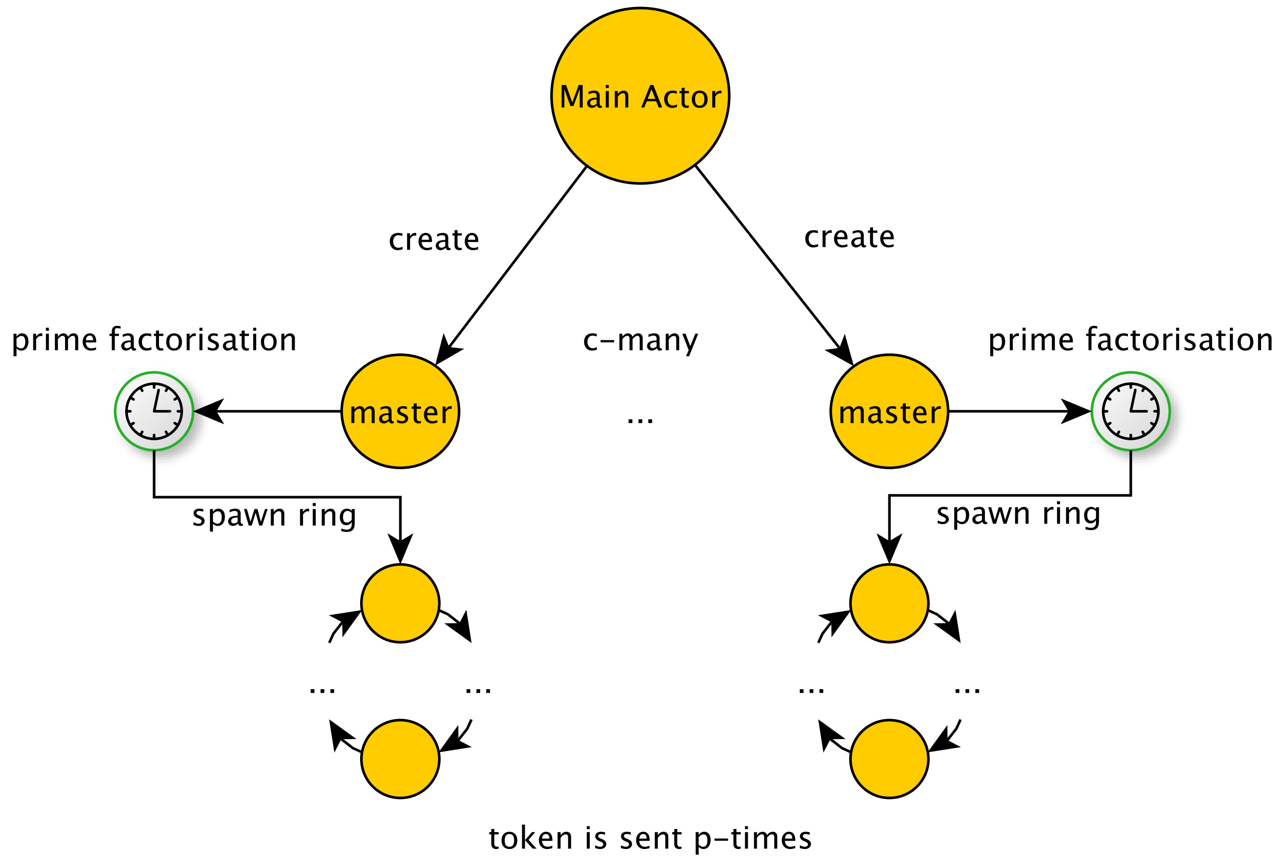
- Nodes acknowledge the termination attempt *t*.

- If all nodes have acknowledged termination without a single node having sent “idle”, the detector sends a termination message to its children (which delegate the message). Otherwise, the termination attempt is cancelled.

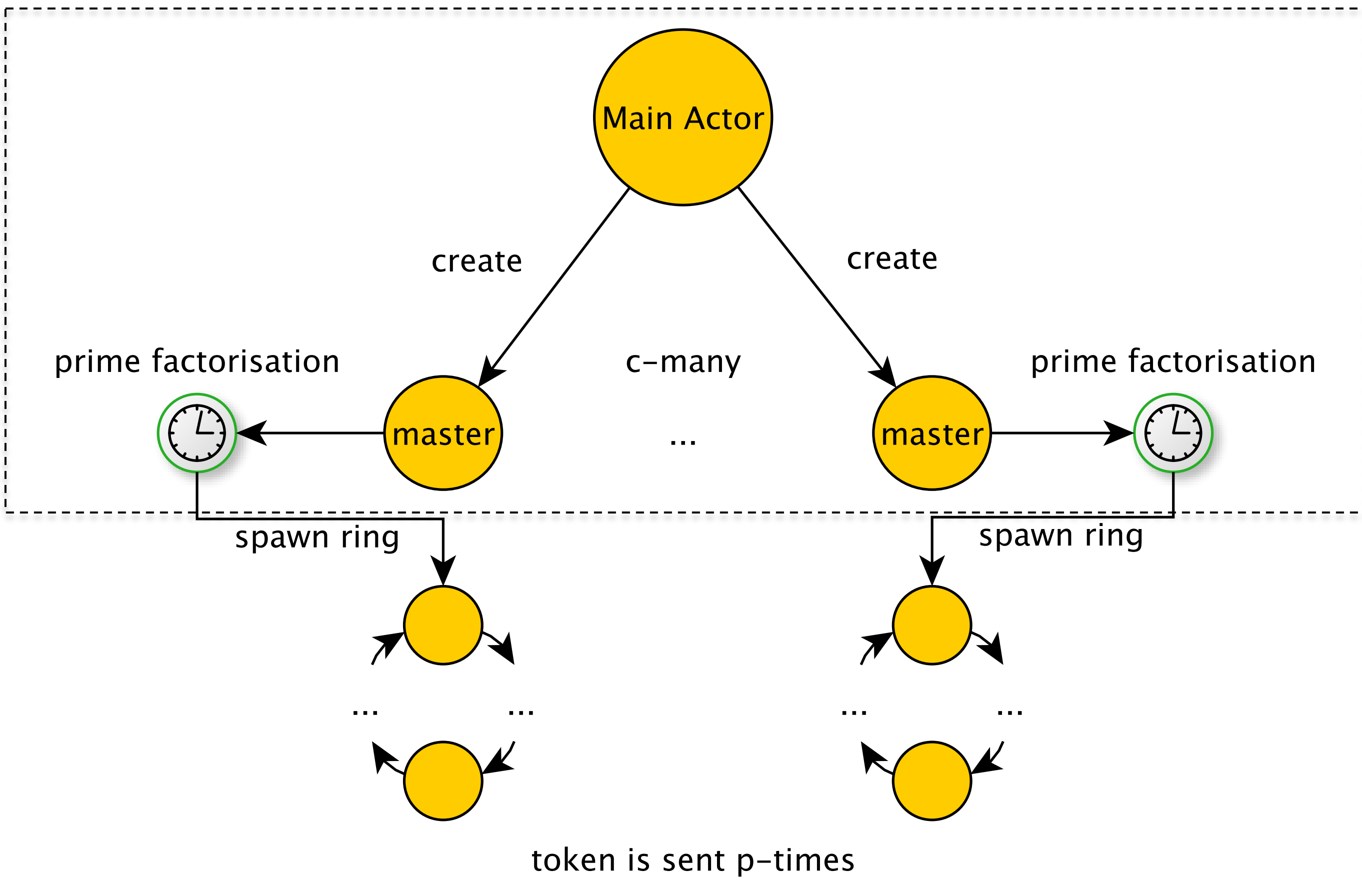
Evaluation

Pony vs. Erlang

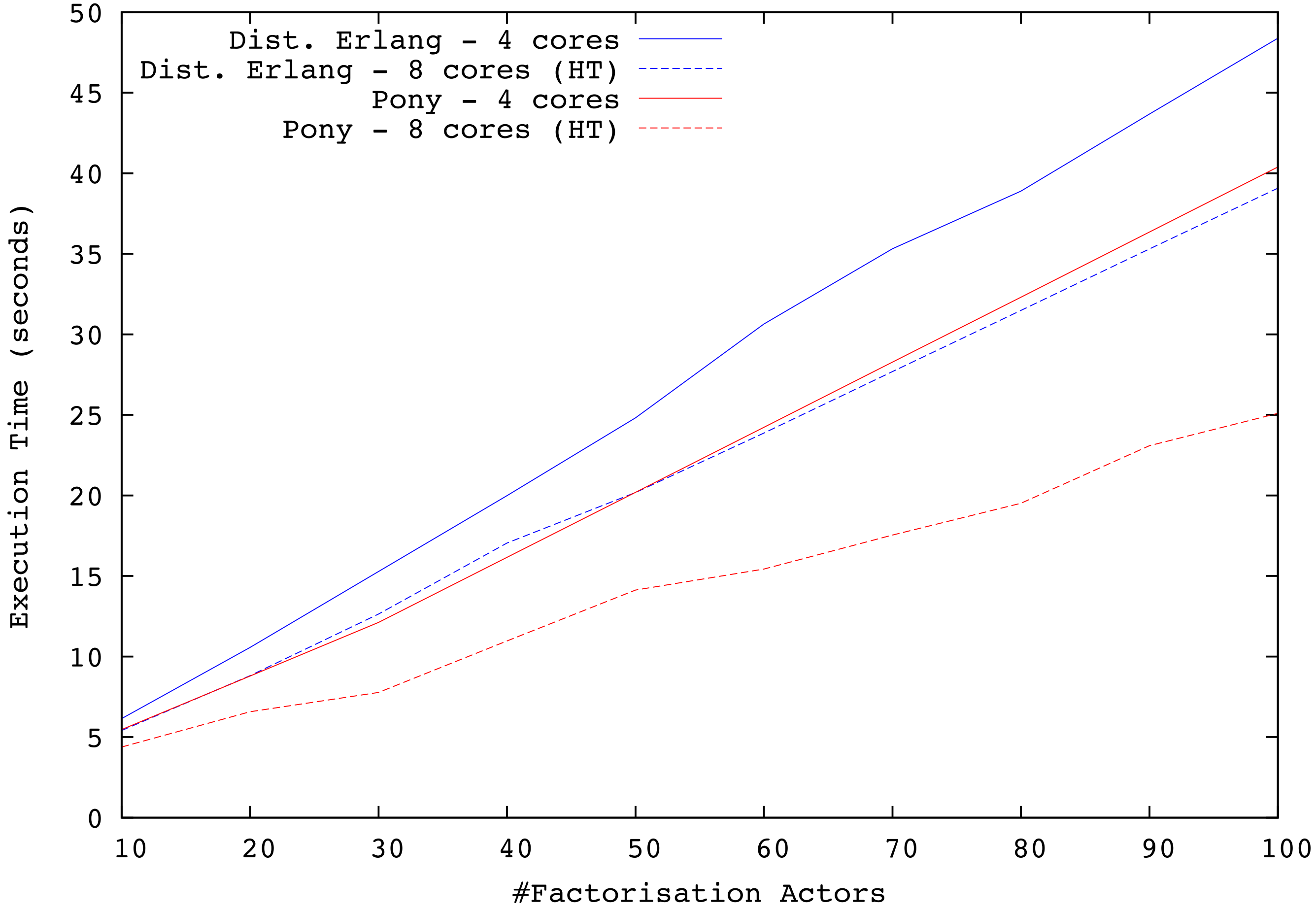
Micro Benchmark



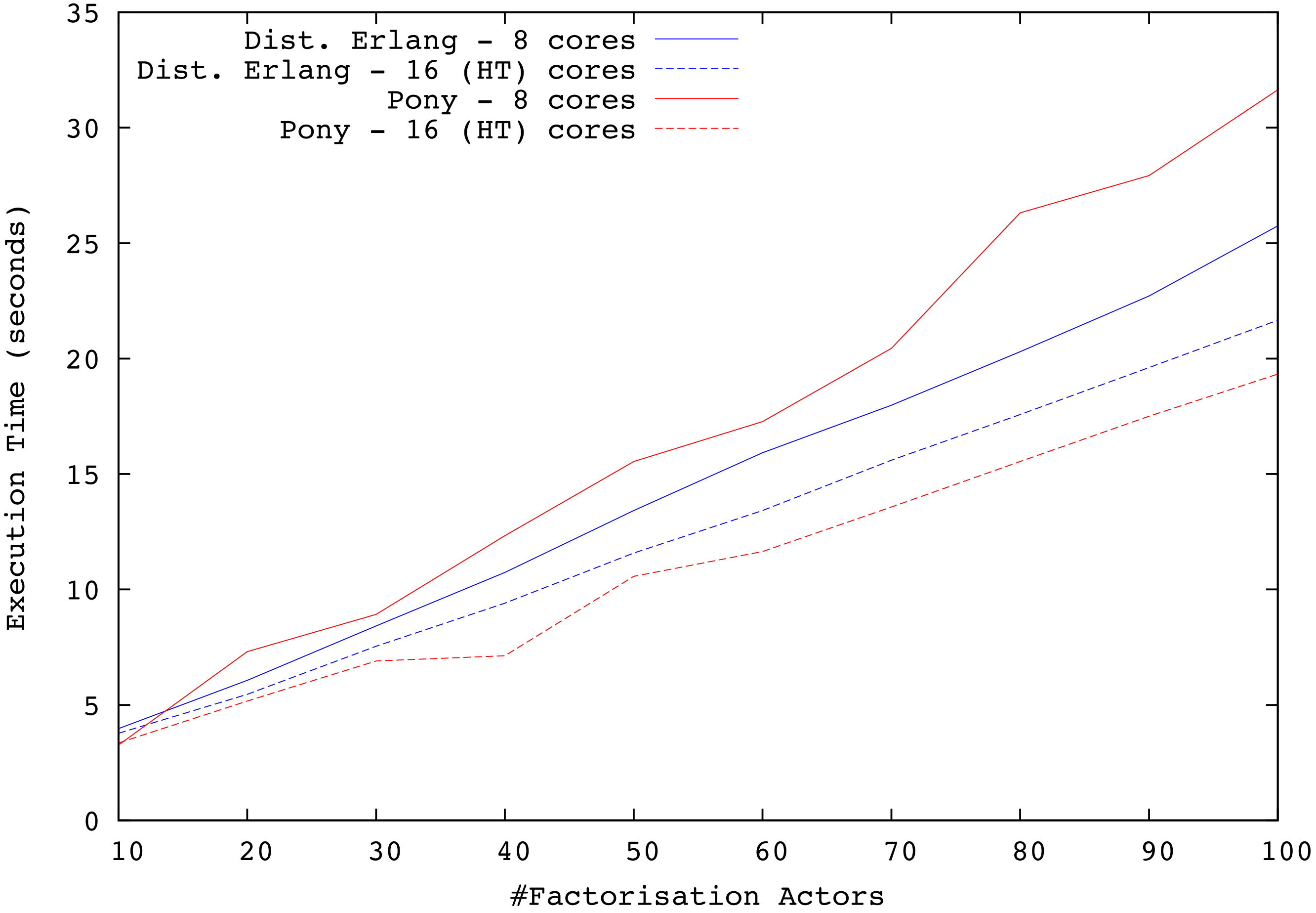
Computation-Boundness



Concurrent Setting

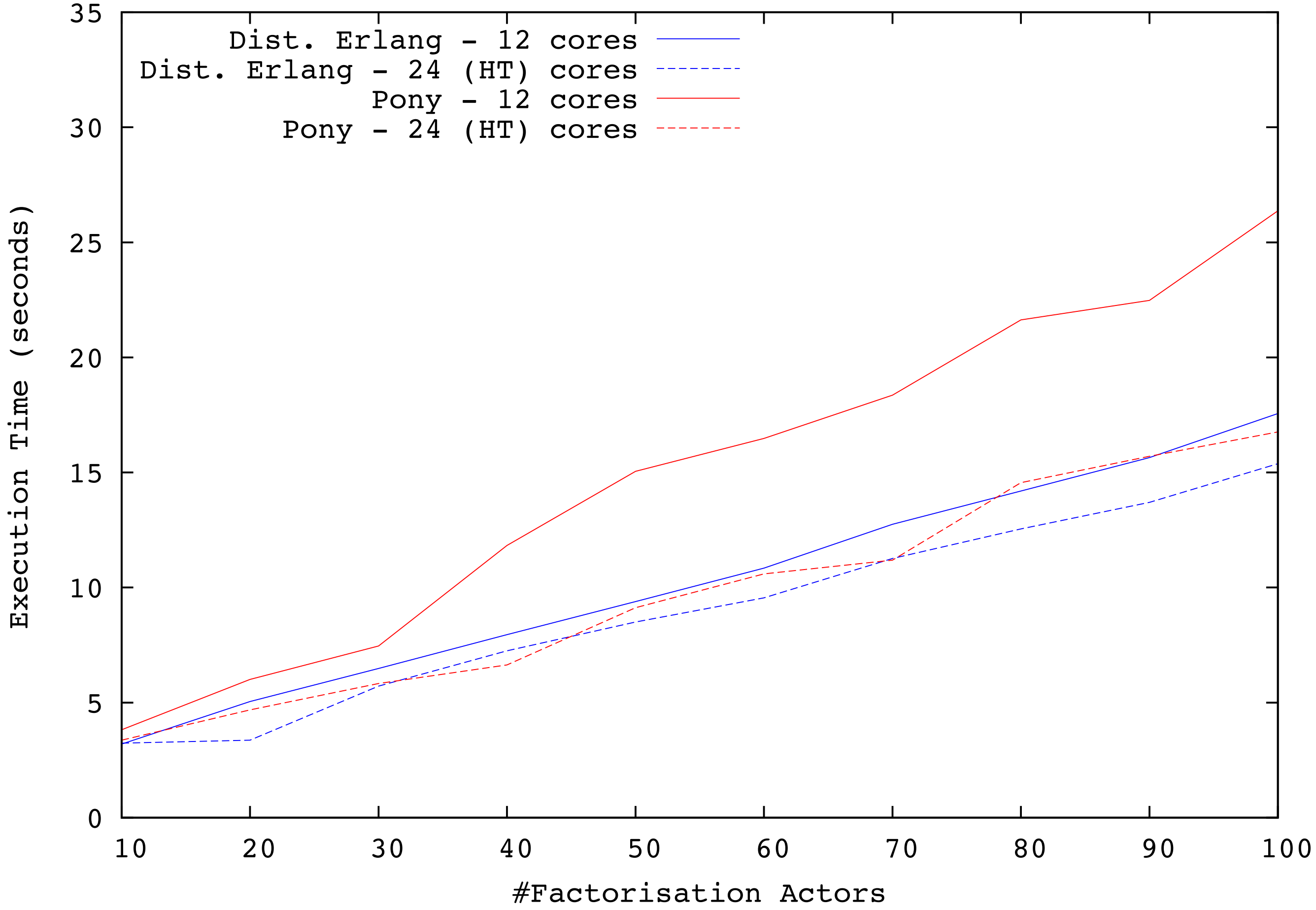


2 nodes



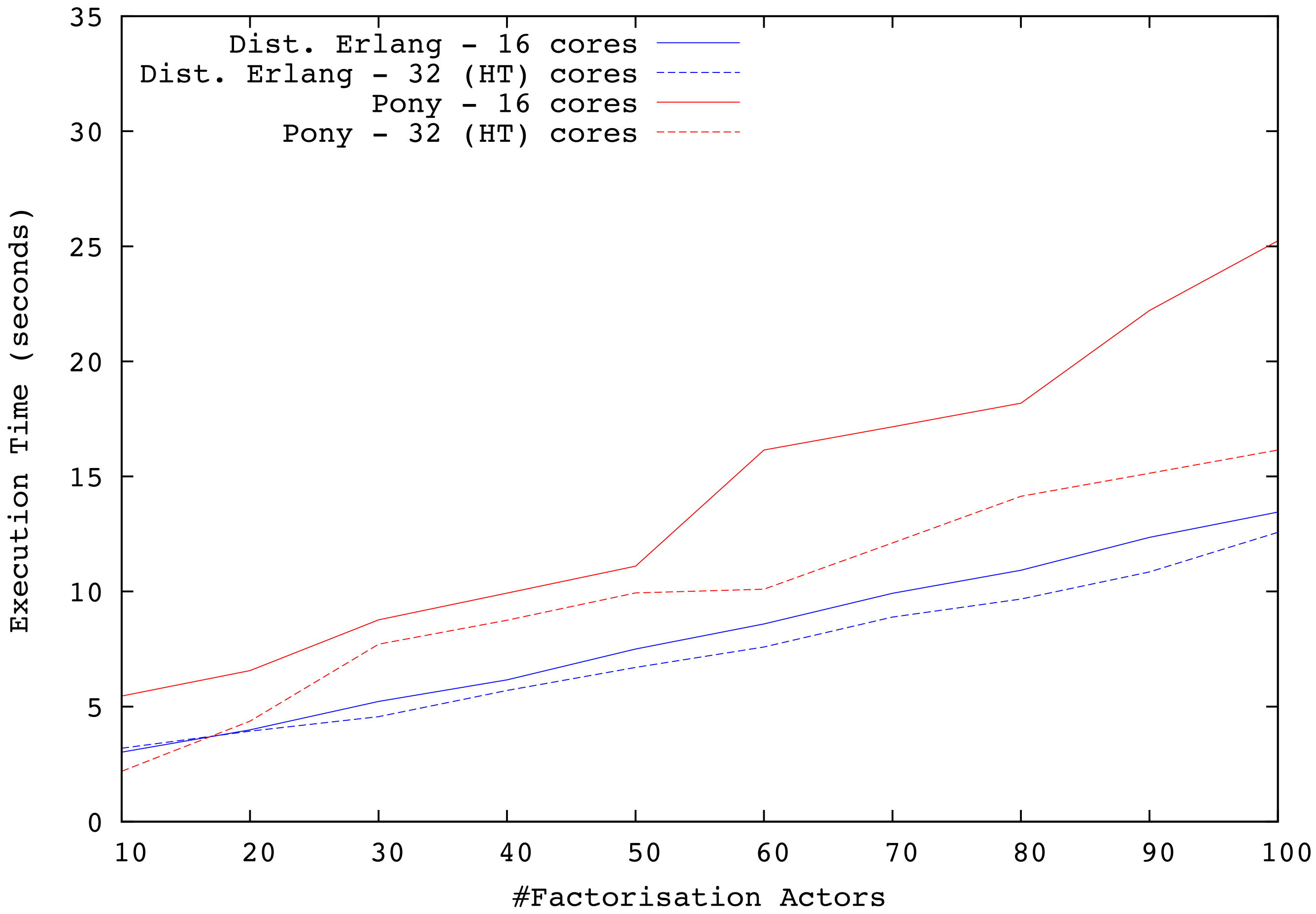
Dist. Erlang - 8 cores
Dist. Erlang - 16 (HT) cores
Pony - 8 cores
Pony - 16 (HT) cores
Execution Time (seconds)
0
5
10
15
20
25
30
35
10
20
30
40
50
60
70
80
90
100
#Factorisation Actors

3 nodes



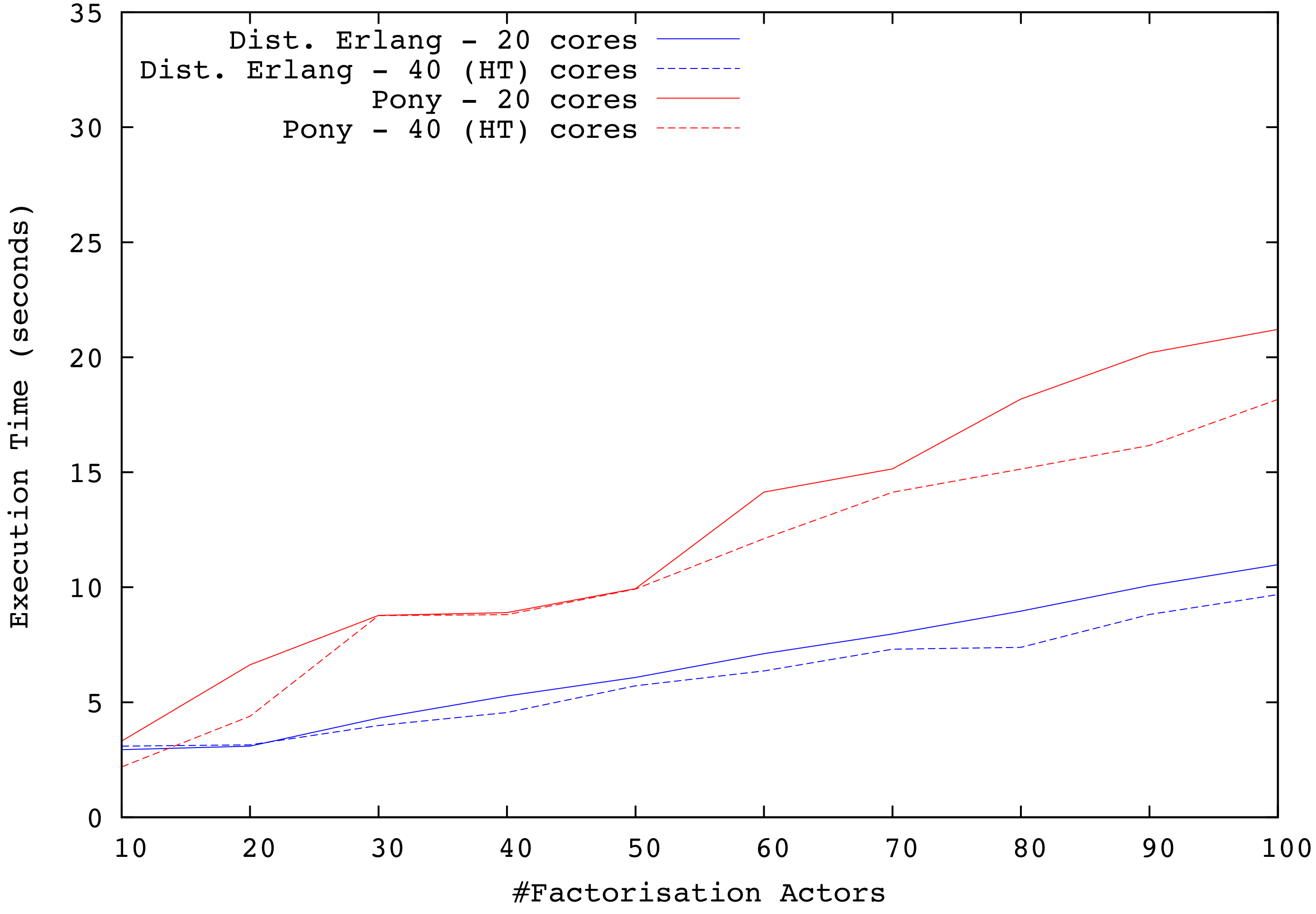
Dist. Erlang - 12 cores
Dist. Erlang - 24 (HT) cores
Pony - 12 cores
Pony - 24 (HT) cores
Execution Time (seconds)
#Factorisation Actors
0
5
10
15
20
25
30
35
10
20
30
40
50
60
70
80
90
100

4 nodes



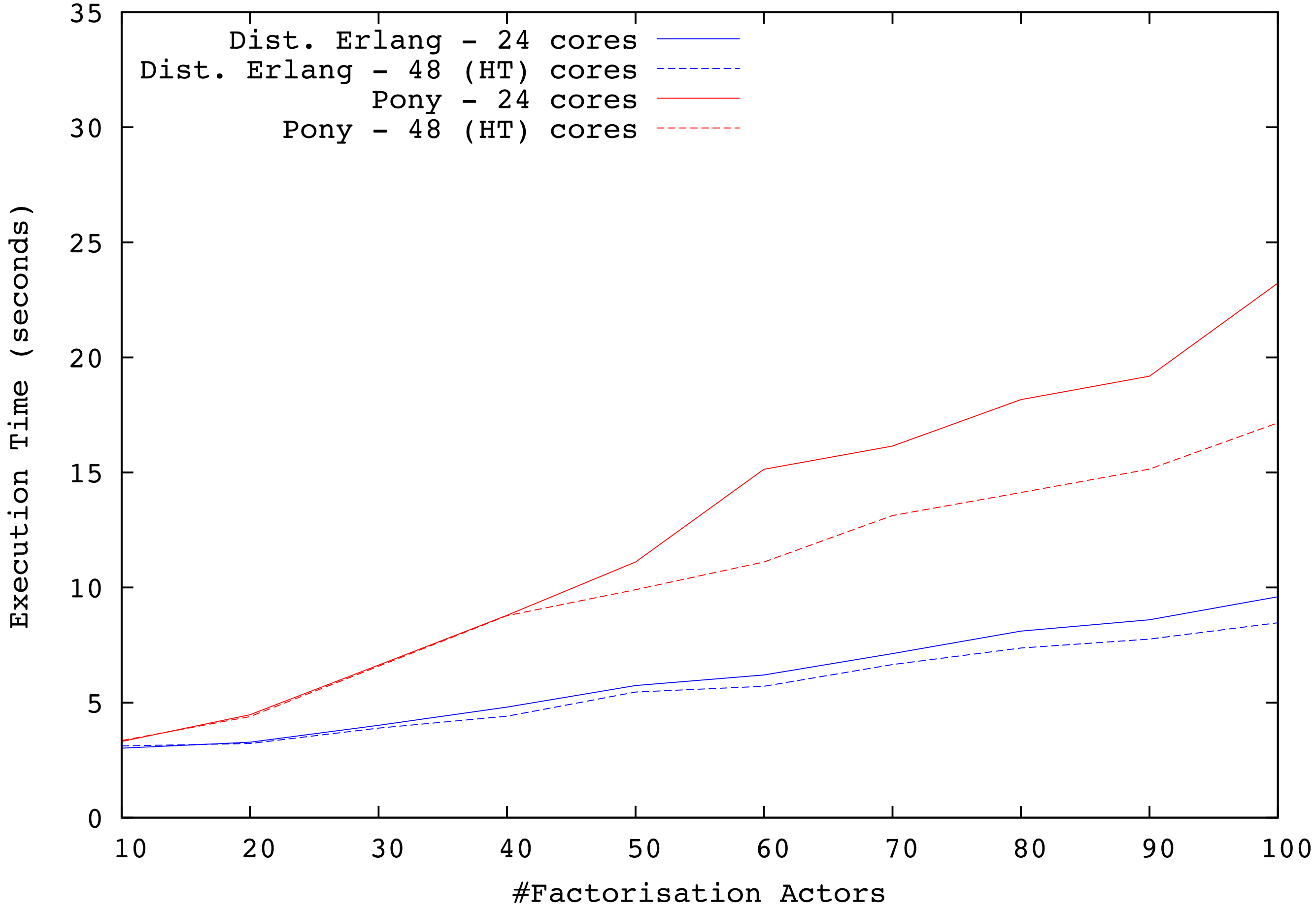
Dist. Erlang - 16 cores
Dist. Erlang - 32 (HT) cores
Pony - 16 cores
Pony - 32 (HT) cores
Execution Time (seconds)
#Factorisation Actors
0
5
10
15
20
25
30
35
10
20
30
40
50
60
70
80
90
100

5 nodes



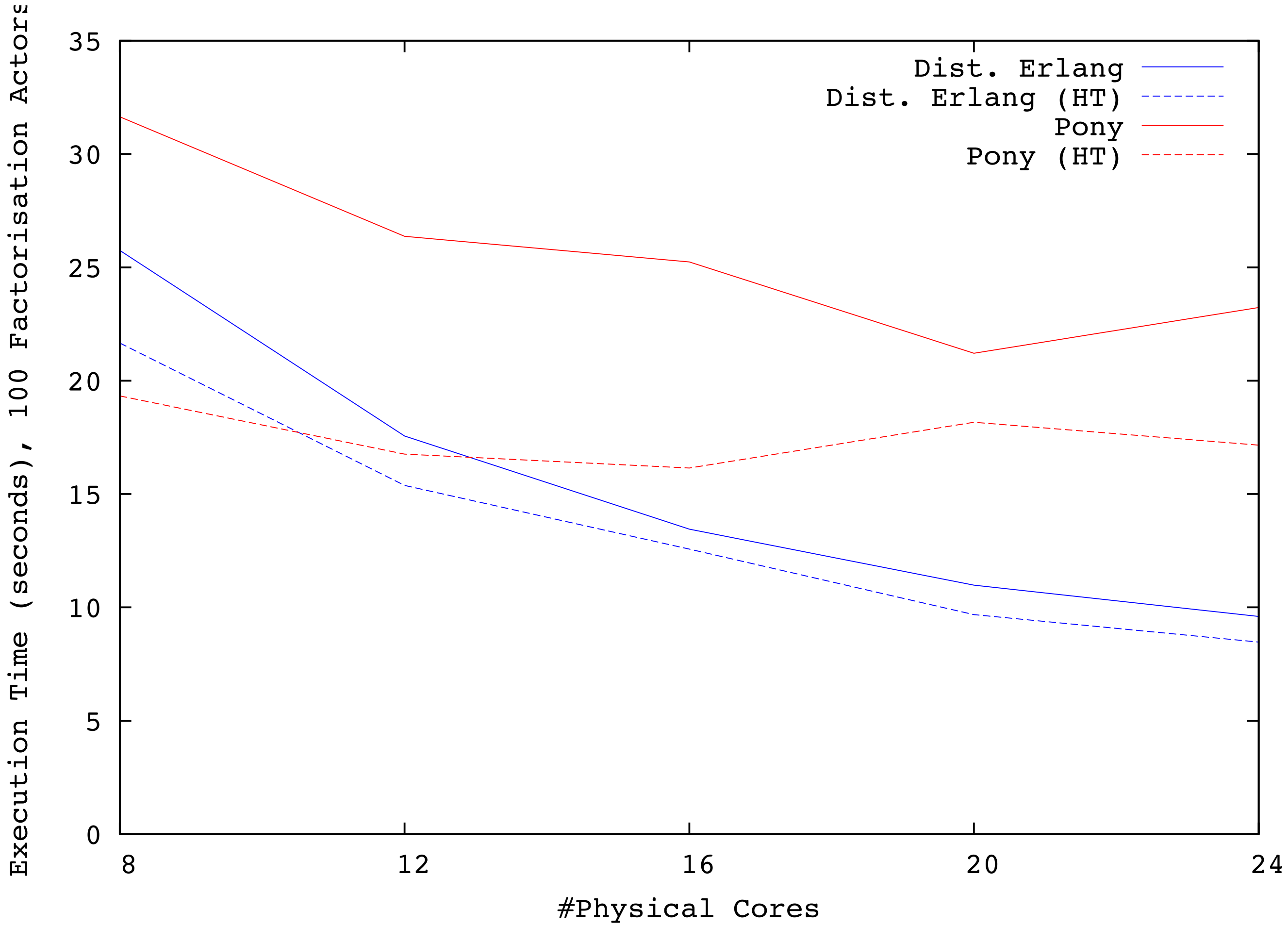
Dist. Erlang - 20 cores
Dist. Erlang - 40 (HT) cores
Pony - 20 cores
Pony - 40 (HT) cores
Execution Time (seconds)
#Factorisation Actors
0
5
10
15
20
25
30
35
10
20
30
40
50
60
70
80
90
100

6 nodes



Dist. Erlang - 24 cores
Dist. Erlang - 48 (HT) cores
Pony - 24 cores
Pony - 48 (HT) cores
Execution Time (seconds)
#Factorisation Actors
0
5
10
15
20
25
30
35
10
20
30
40
50
60
70
80
90
100

Trend



Execution Time (seconds), 100 Factorisation Actors
35
30
25
20
15
10
5
0
8
12
16
20
24
#Physical Cores
Dist. Erlang
Dist. Erlang (HT)
Pony
Pony (HT)

1000 Factorisation Actors

Nodes	Dist. Erlang	Dist.Erlang (HT)	Pony	Pony (HT)
1	473.85s	380.55s	393.39s	244.35s
2	241.45s	205.43s	321.90s	184.98s
3	165.45s	142.70s	276.37s	157.22s
4	126.11s	115.62s	239.23s	143.14s
5	101.91s	87.53s	223.22s	141.13s
6	86.76s	75.07s	225.25s	142.16s

1 node = 1x Intel i7 – Quad Core @ 3.40 GHz HT, 8 GB Memory

Future Work

- Improve I/O Multiplexing
- Failure Detection and Dynamic Tree Topology
- Locality-aware Tree Topology
- Tune Distributed Work Stealing and Migration
- Compiler, Type System, Formal Models
- Generalisation

- Publications

Backup Slides

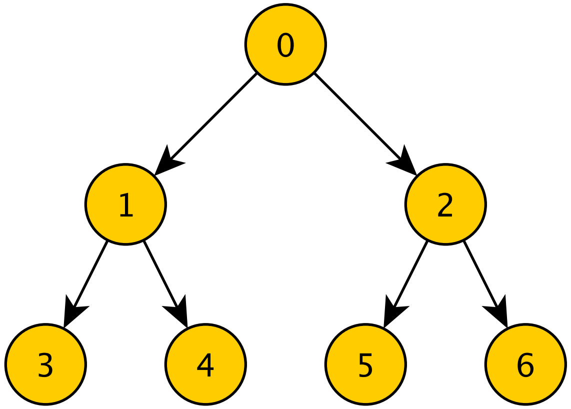
Joining new Slave Nodes

- Adding new nodes at runtime
- Configure Tree Topology

- Challenges:
  - No need for maintaining global view at each node
  - Nodes should not be required to store any information in order to determine the path via which a new slave should be delegated
  - Guarantee that the tree is *almost* balanced at any point in time

- Answer: Find a location of a new node based on its ordinal number

Joining new Slave Nodes



Joining Algorithm

$$All(d) = \sum_{i=1, i \in \mathbb{N}}^{d} k^i$$

$$Ord : \mathbb{N}^* \rightarrow \mathbb{N}$$
$$Ord(a_1 ... a_d) = All(d-1) + (((a_1 * k + a_2) * k + a_3)) * k + ... + a_{d-1}) * k + a_d + 1$$

$$Ord^{-1} : \mathbb{N}^* \rightarrow \mathbb{N}$$
$$\forall m \in \mathbb{N}^+ : Ord^{-1}(m) = a_1 ... a_n \text{ such that } \forall i \in \mathbb{N}^+ : 0 < a_i \leq k \wedge Ord(a_1 ... a_n) = m$$

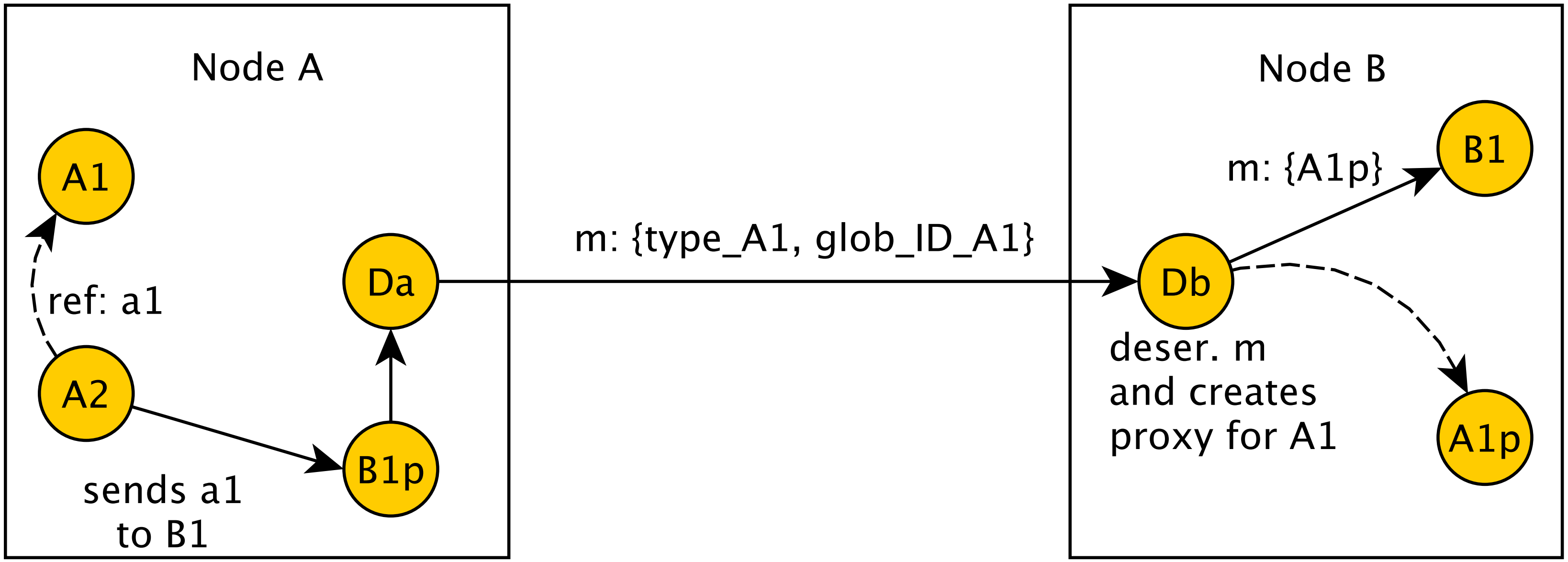
$$b_p = m - All(p-1) - 1$$

for i in range(p, 1):

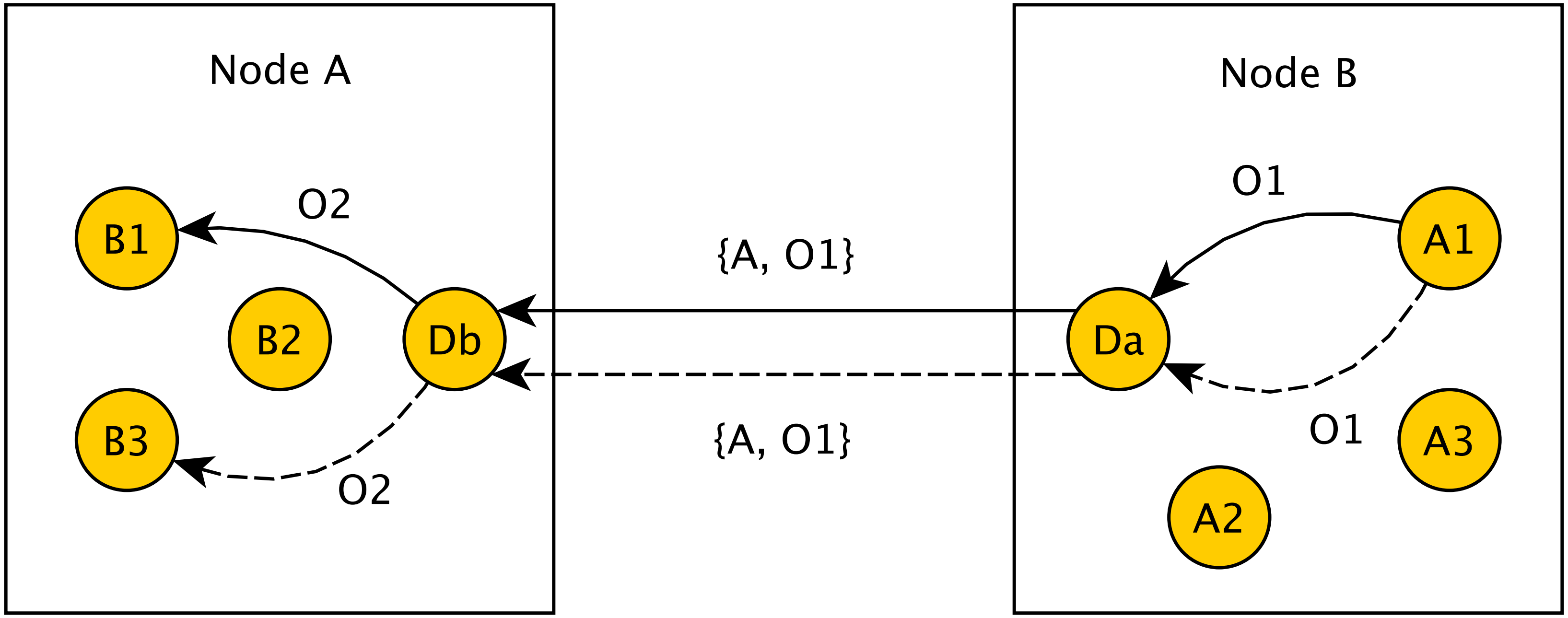
$$a_i := b_i \bmod k$$

$$b_{i-1} := \frac{(b_i - a_i)}{k}$$

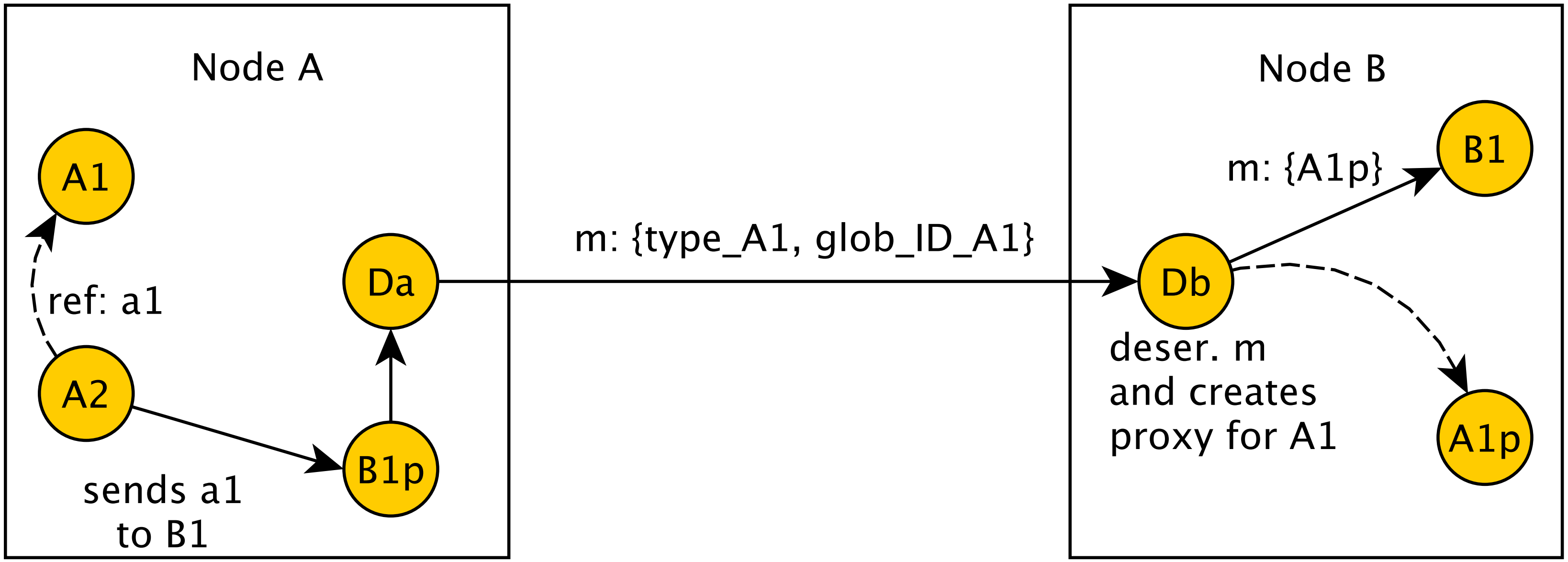
Handling Actor References



Object Identity Comparison



Distributed Reference Counting (Actors)



Distributed Reference Counting (Objects)