Resilient Optimistic Termination Detection for the Async-Finish Model

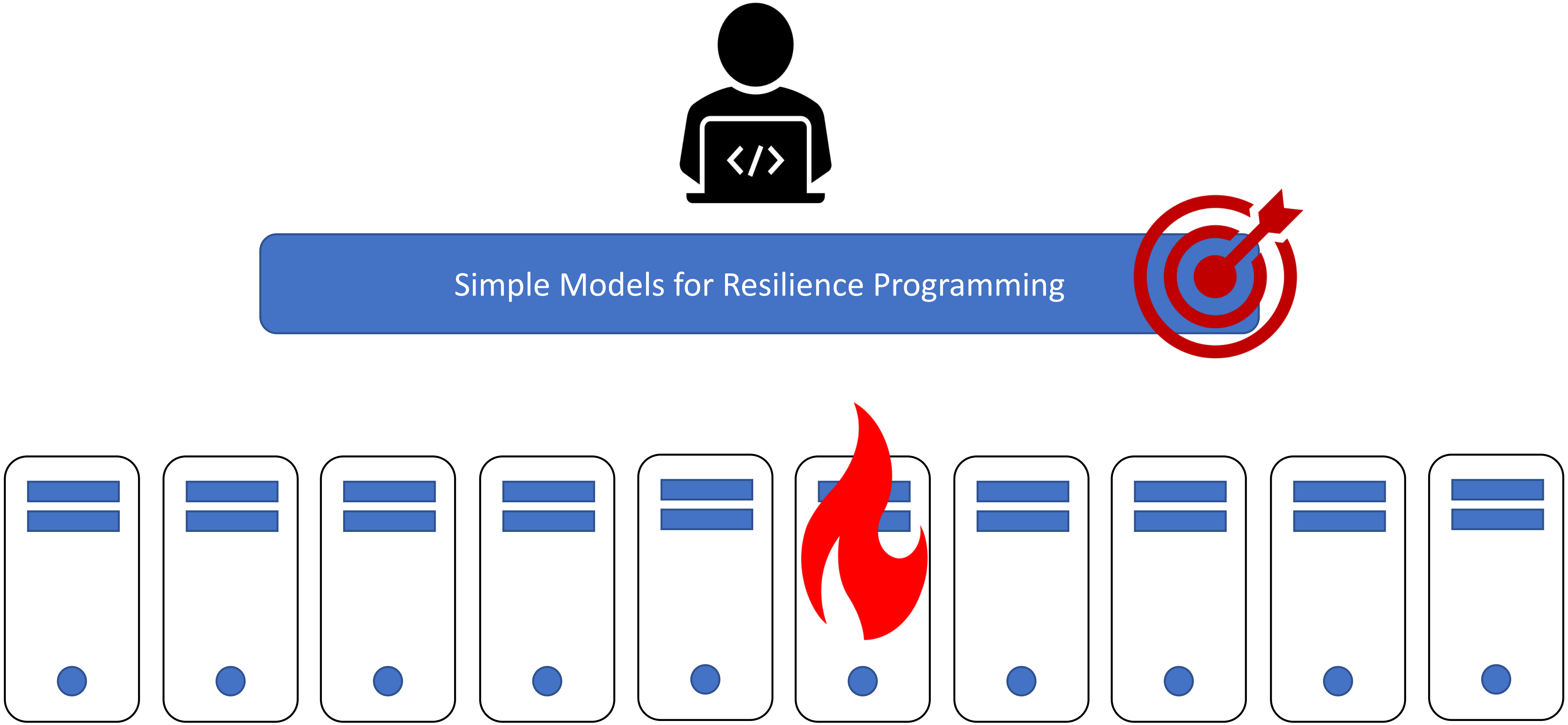
Sara S. Hamouda[1,2] and Josh Milthorpe[1]

[1]Australian National University, Australia

[2]Inria, France

ISC-HPC 2019

Research Aim



Simple Models for Resilience Programming

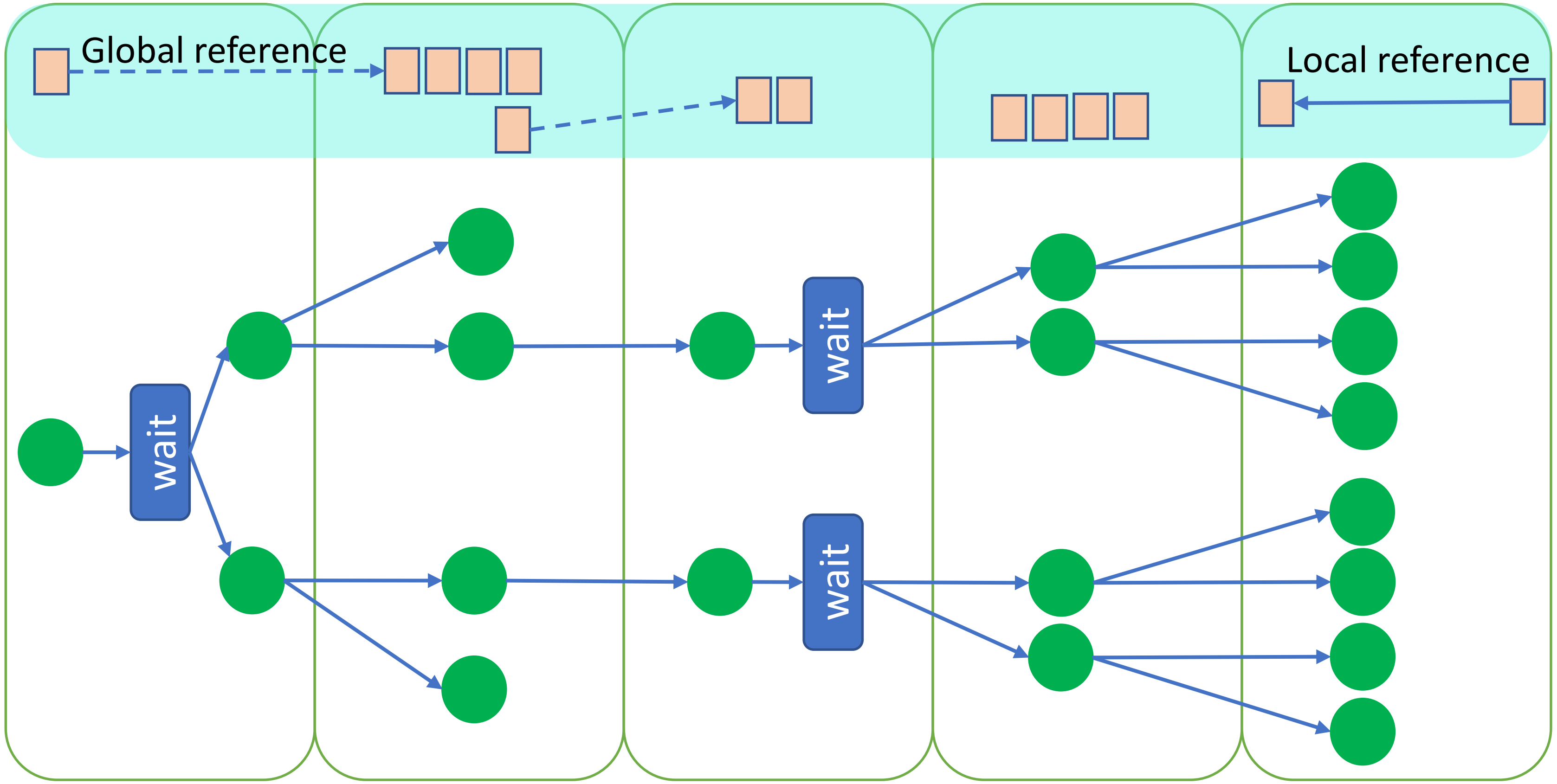
Asynchronous Partitioned Global Address Space Model

Examples: **X10** (from IBM) and **Chapel** (from Cray)

Global Address Space

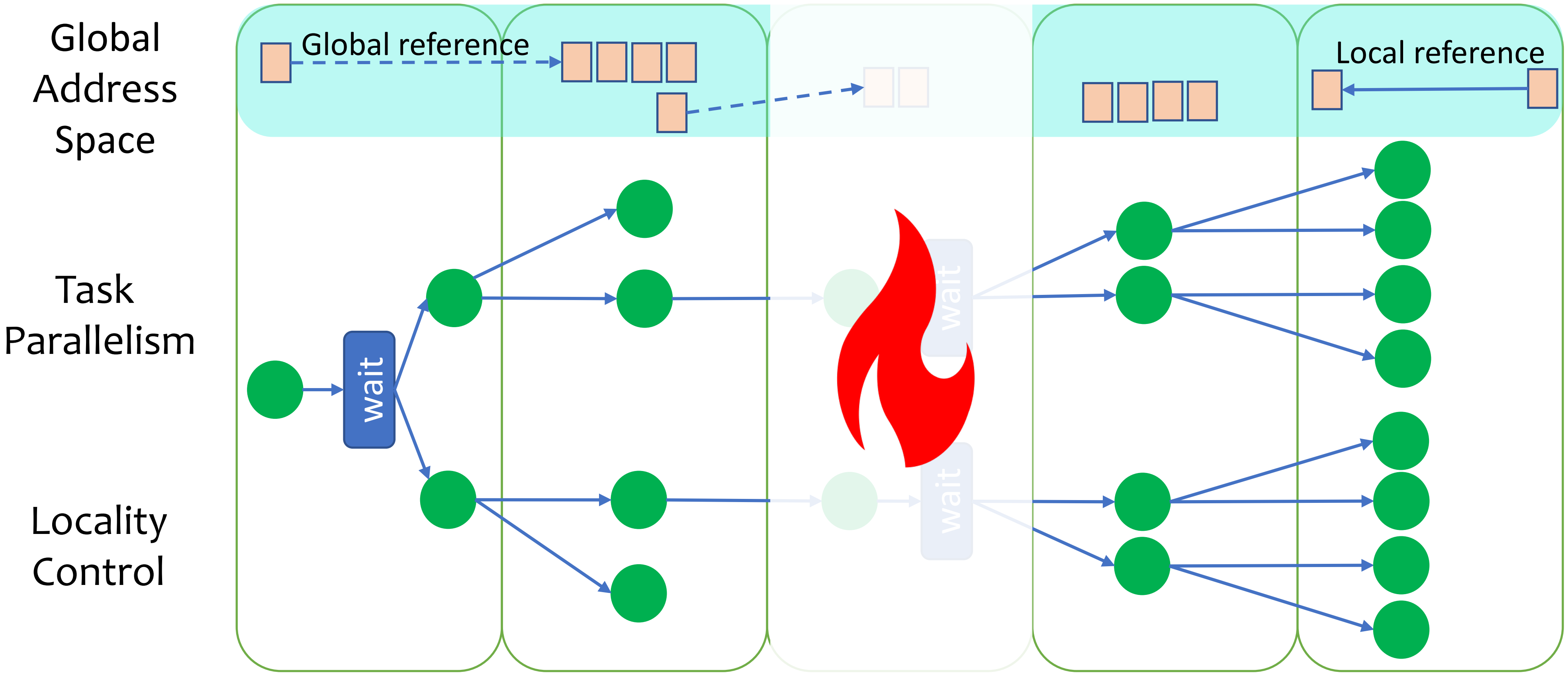
Task Parallelism

Locality Control



Asynchronous Partitioned Global Address Space Model

Examples: **X10** (from IBM) and **Chapel** (from Cray)



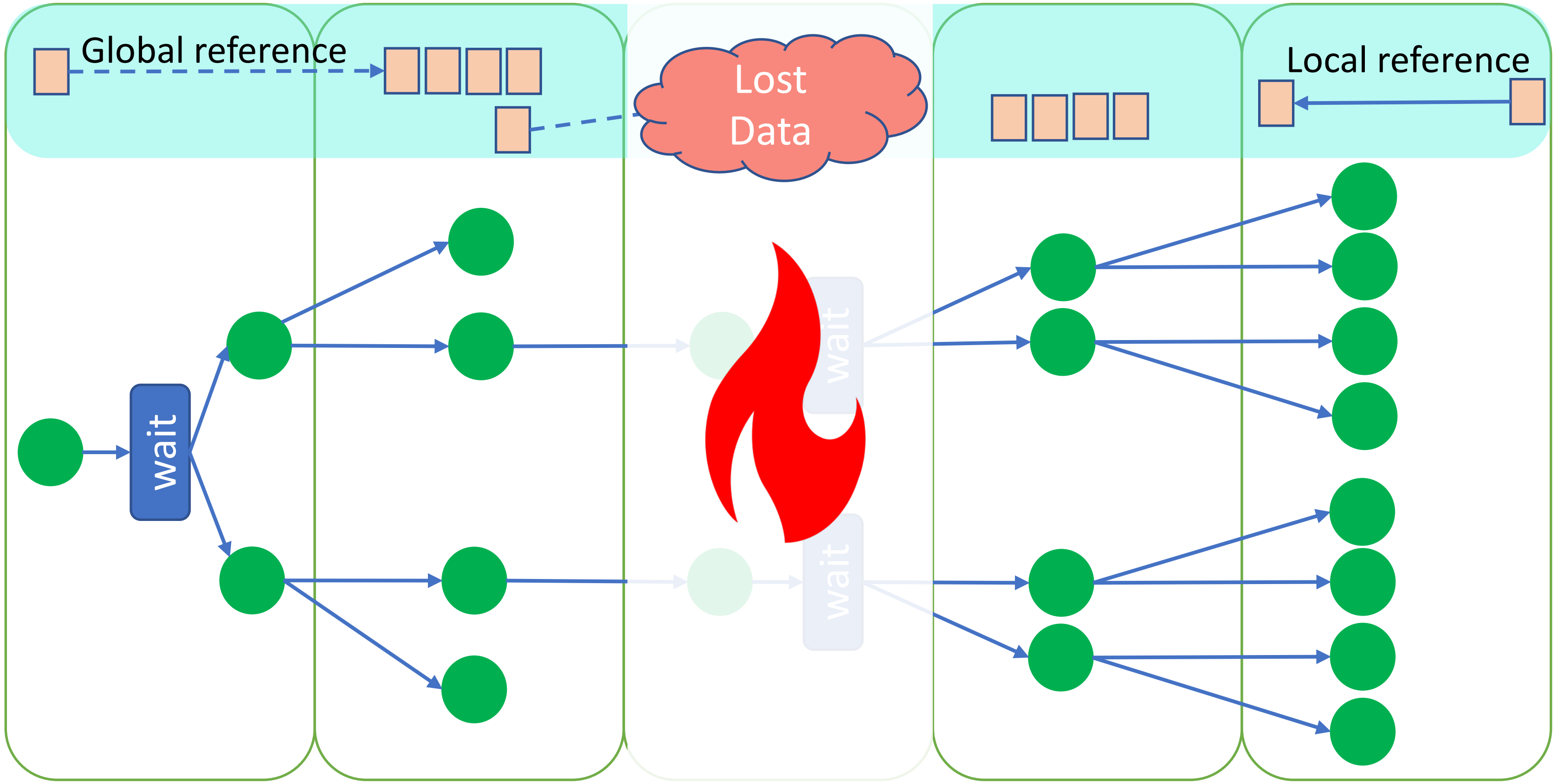
Asynchronous Partitioned Global Address Space Model

Examples: **X10** (from IBM) and **Chapel** (from Cray)

Global Address Space

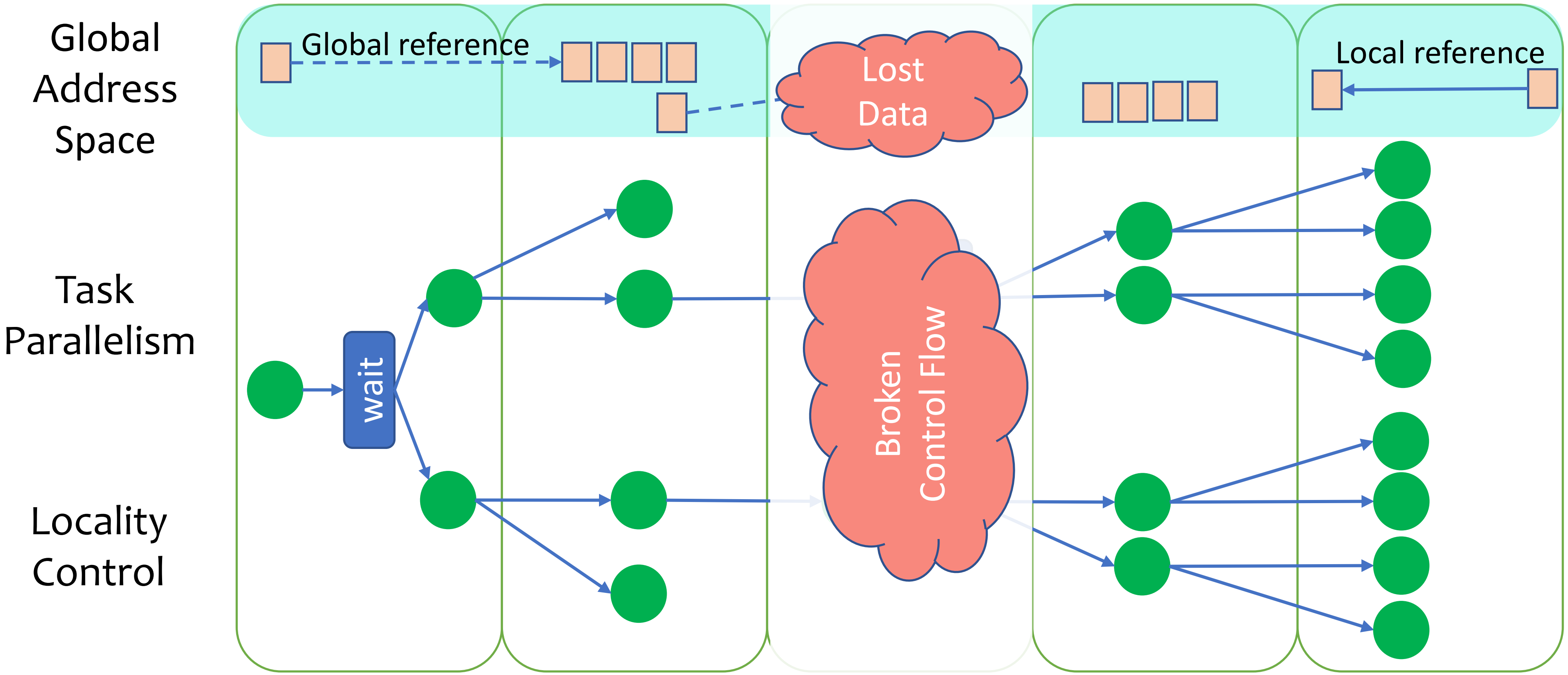
Task Parallelism

Locality Control

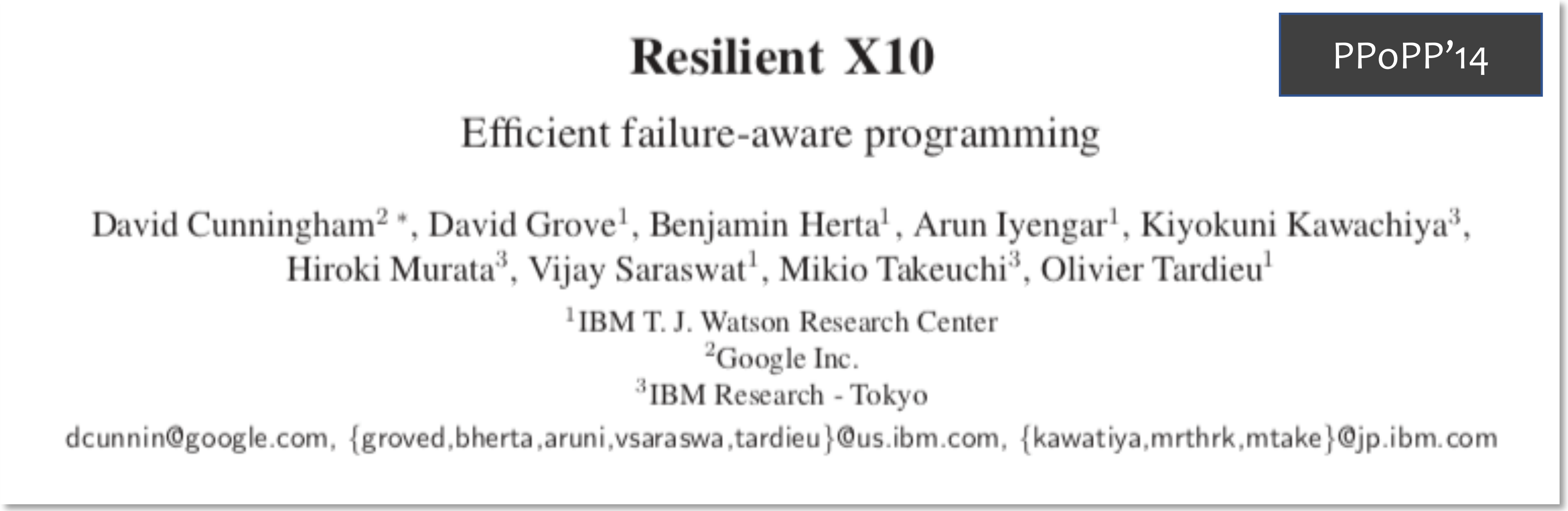


Asynchronous Partitioned Global Address Space Model

Examples: **X10** (from IBM) and **Chapel** (from Cray)



Resilient X10



PPoPP’14

Resilient X10

Efficient failure-aware programming

David Cunningham[2] *, David Grove[1], Benjamin Herta[1], Arun Iyengar[1], Kiyokuni Kawachiya[3],
Hiroki Murata[3], Vijay Saraswat[1], Mikio Takeuchi[3], Olivier Tardieu[1]

[1] IBM T. J. Watson Research Center
[2] Google Inc.
[3] IBM Research - Tokyo

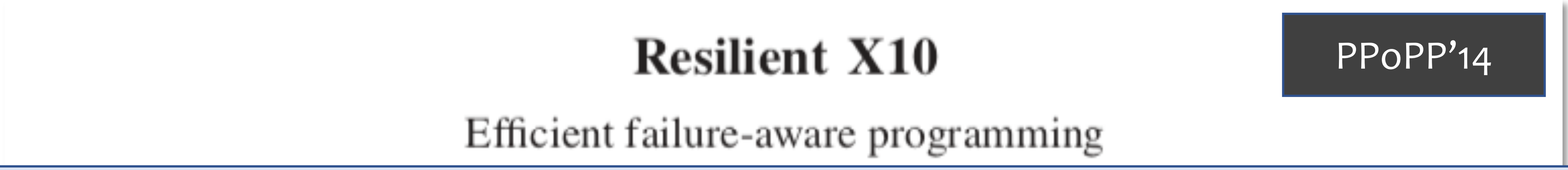
dcunnin@google.com, {groved,bherta,aruni,vsaraswa,tardieu}@us.ibm.com, {kawatiya,mrthrk,mtake}@jp.ibm.com

Control Flow Repair

Resilient Termination Detection Protocol

Data Recovery

Resilient X10



Resilient X10

Efficient failure-aware programming

PPoPP’14

Protocol inefficiencies

- **Pessimistic protocol**
  It favours the simplicity of failure recovery over failure-free performance.
- **Not message-optimal**
  It uses more task tracking messages than strictly required.

Control Flow Repair

Resilient Termination Detection Protocol

Data Recovery

Agenda

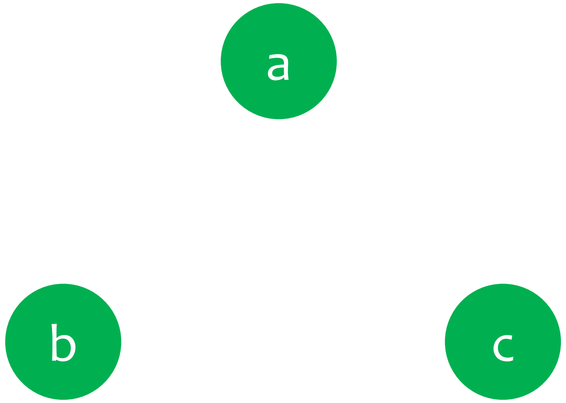
- **Background**
  - The Async-Finish Task Model
- **Async-Finish Termination Detection**
  - The non-resilient protocol
  - The pessimistic protocol
  - The optimistic protocol
- **Performance Evaluation**
  - Microbenchmarks
  - LULESH application

The Async-Finish Task Model

`async`	`finish`	`at`
Task Creation	Synchronization	Locality

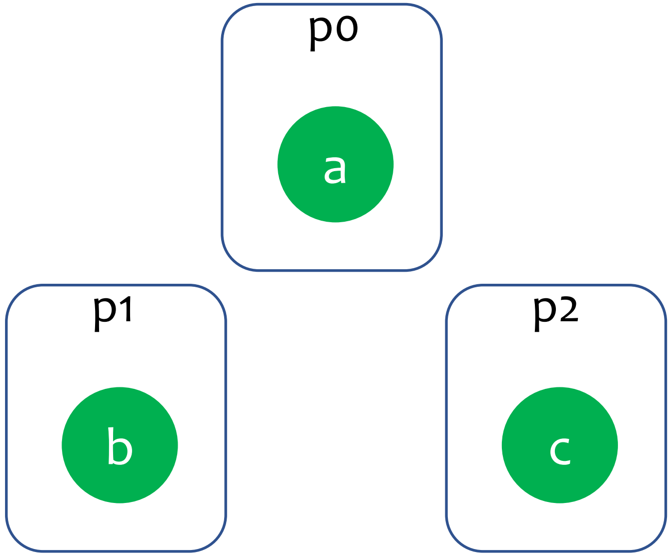
The Async-Finish Task Model

```
async { /*a*/ }

async { /*b*/ }

async { /*c*/ }
```



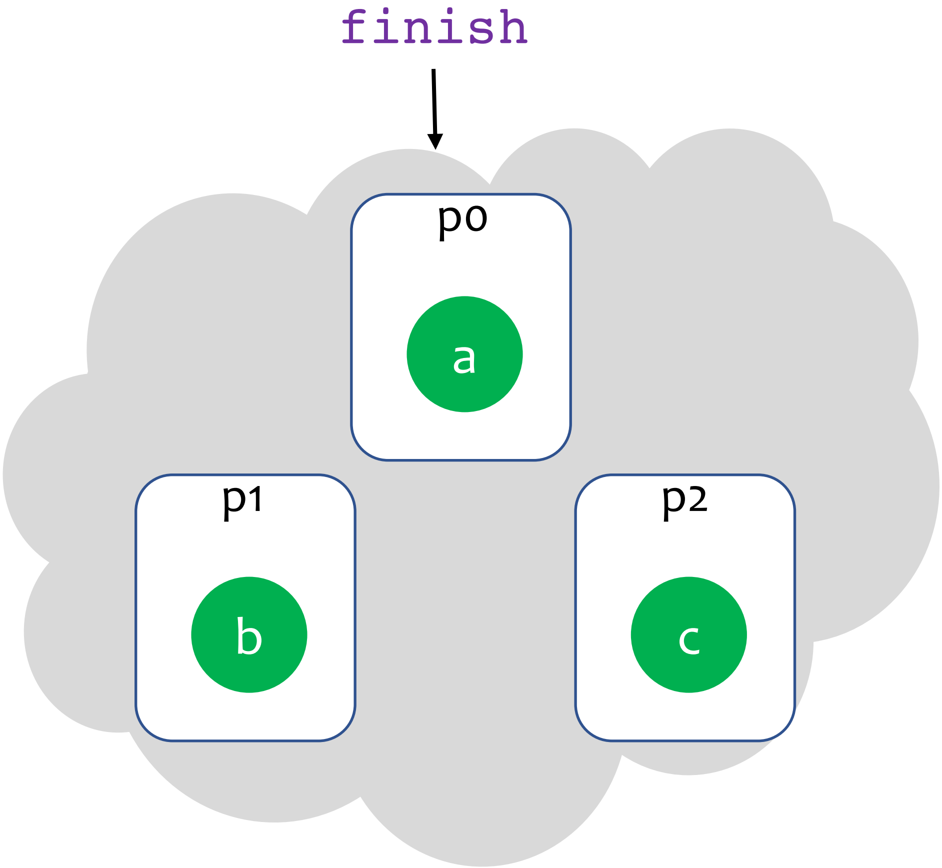
The Async-Finish Task Model

```
at (p0) async { /*a*/ }

at (p1) async { /*b*/ }

at (p2) async { /*c*/ }
```



The Async-Finish Task Model

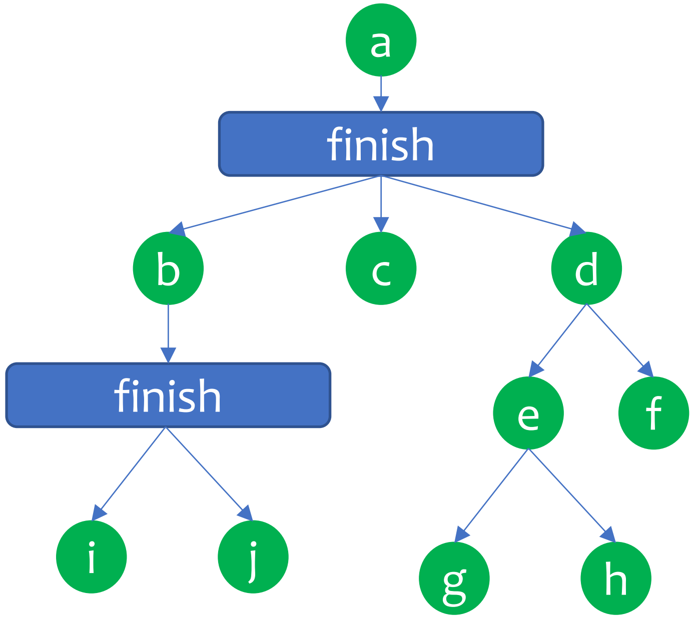
```
finish {

    at (p0) async { /*a*/ }

    at (p1) async { /*b*/ }

    at (p2) async { /*c*/ }

}
Z;
```



Async-Finish versus Spawn-Sync

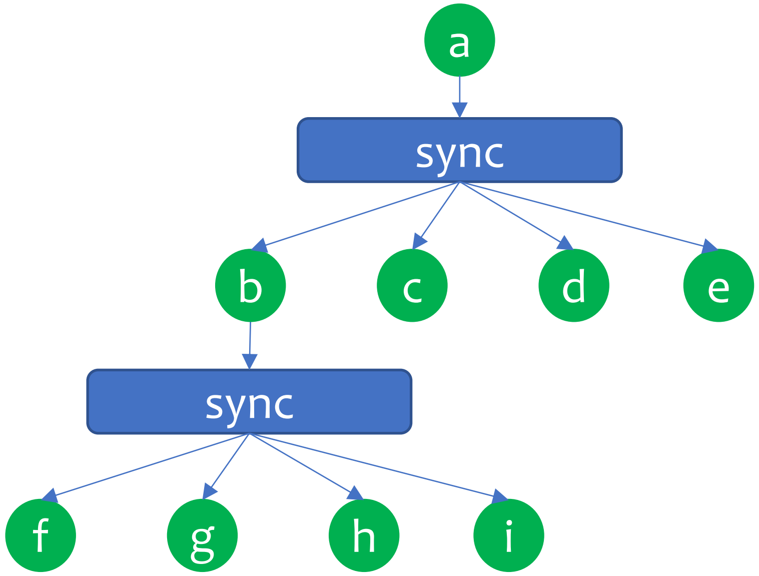
Async-Finish (Terminally-Strict)

- A task can wait for other tasks it directly or transitively spawned.



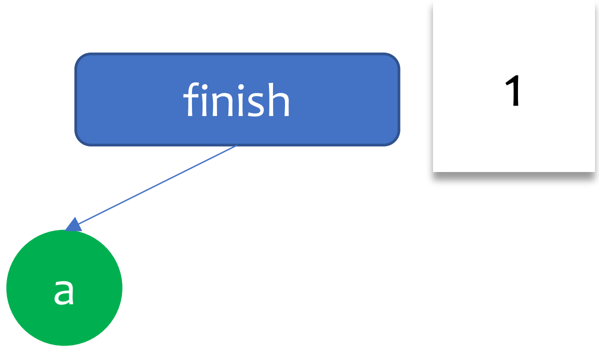
Spawn-Sync (Fully-Strict)

- A task can wait for other tasks it directly spawned.



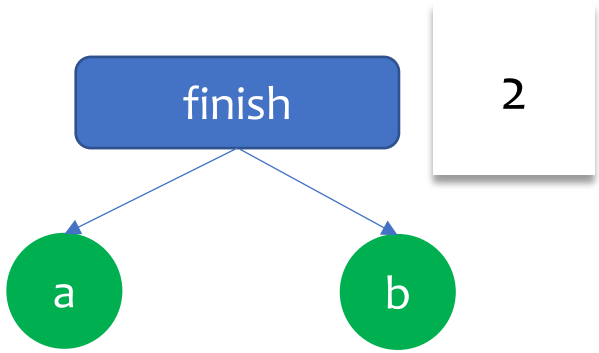
Finish Termination Detection

- Finish tracks the number of active tasks within its scope.



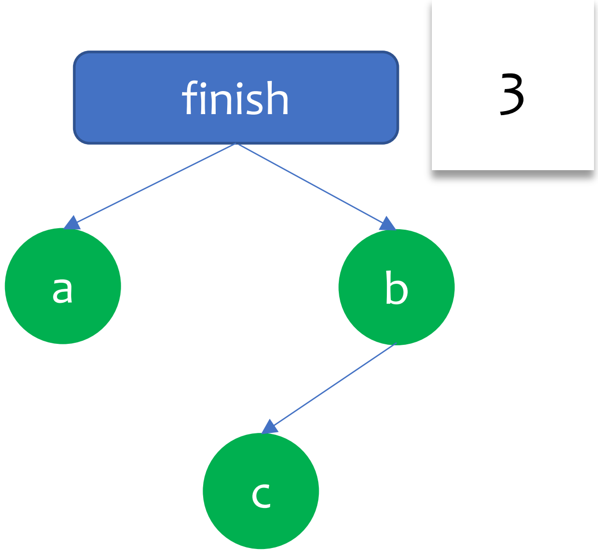
Finish Termination Detection

- Finish tracks the number of active tasks within its scope.



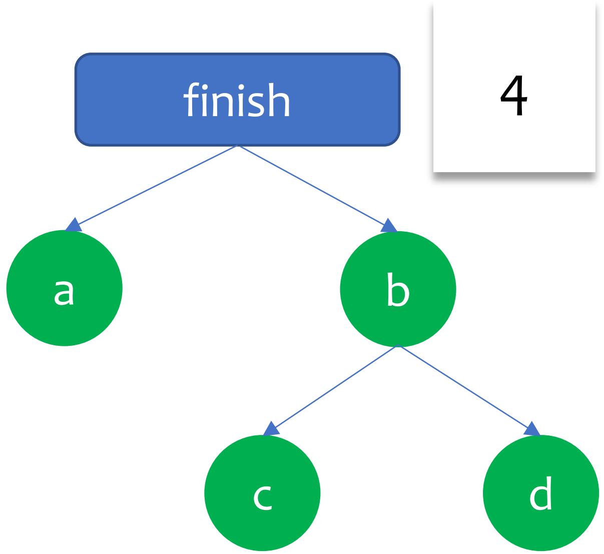
Finish Termination Detection

- Finish tracks the number of active tasks within its scope.



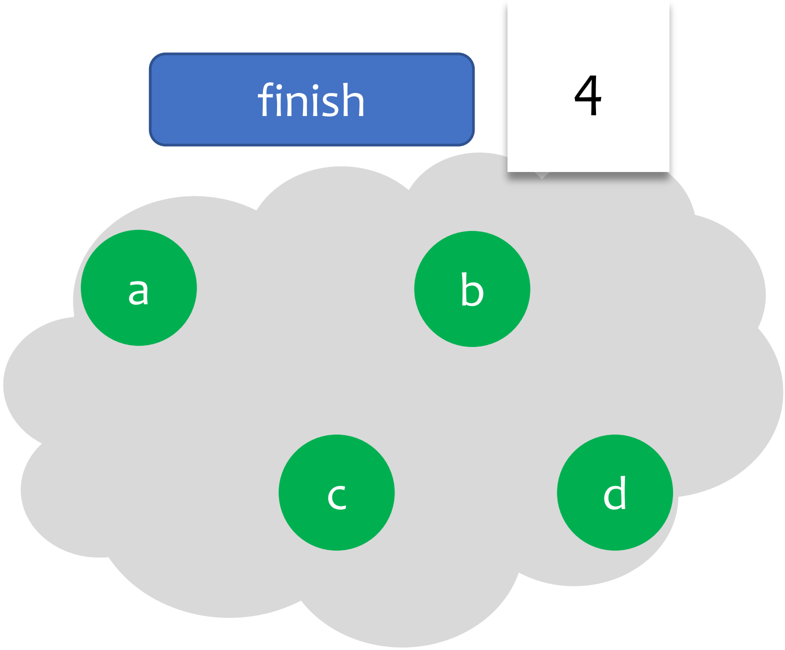
Finish Termination Detection

- Finish tracks the number of active tasks within its scope.



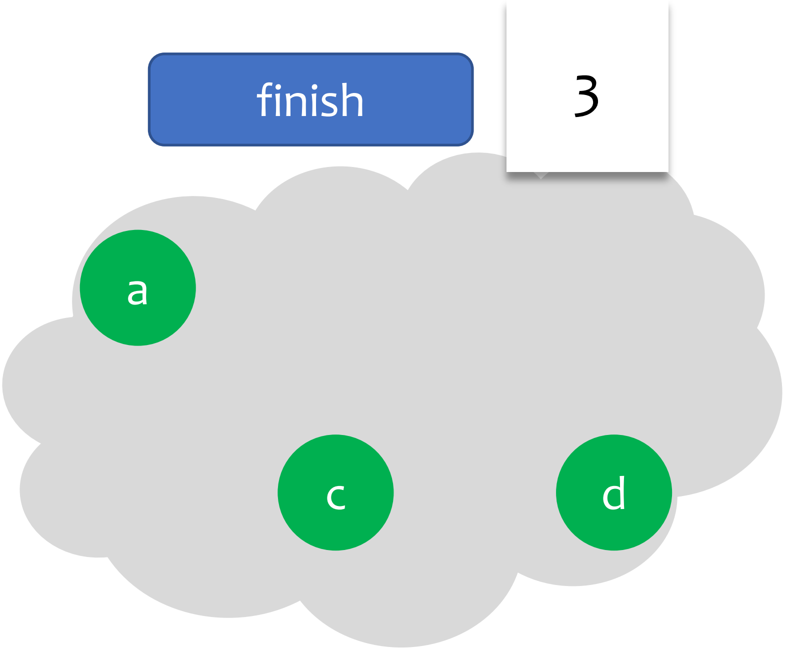
Finish Termination Detection

- Finish tracks the number of active tasks within its scope.



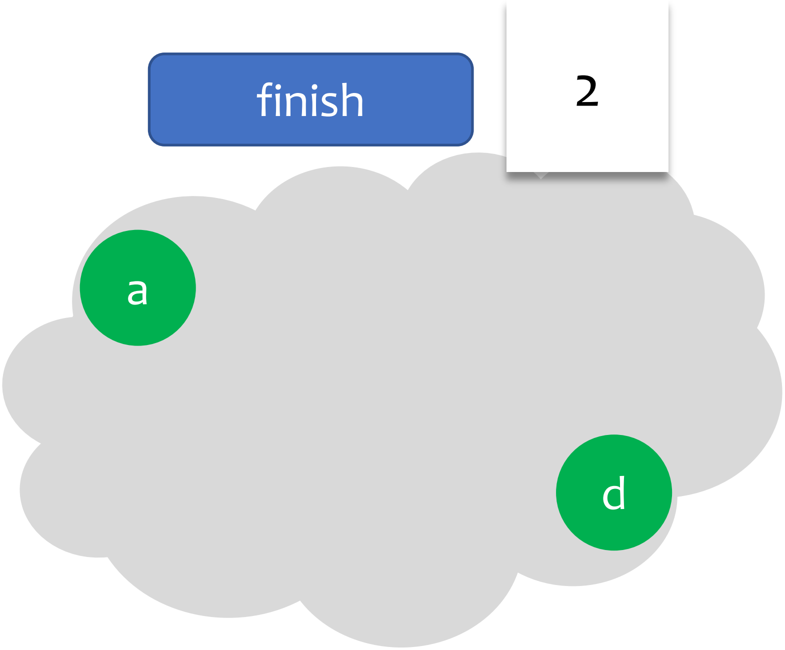
Finish Termination Detection

- Finish tracks the number of active tasks within its scope.



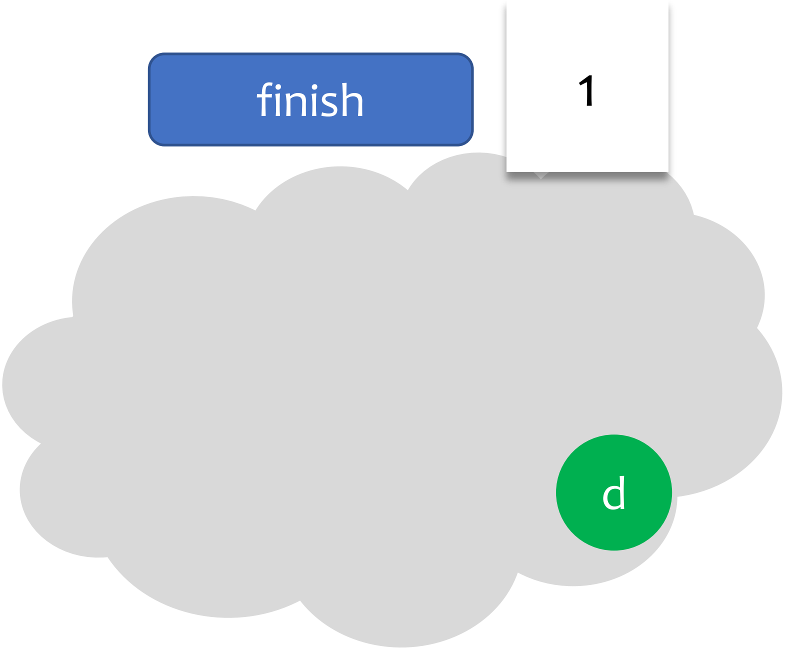
Finish Termination Detection

- Finish tracks the number of active tasks within its scope.



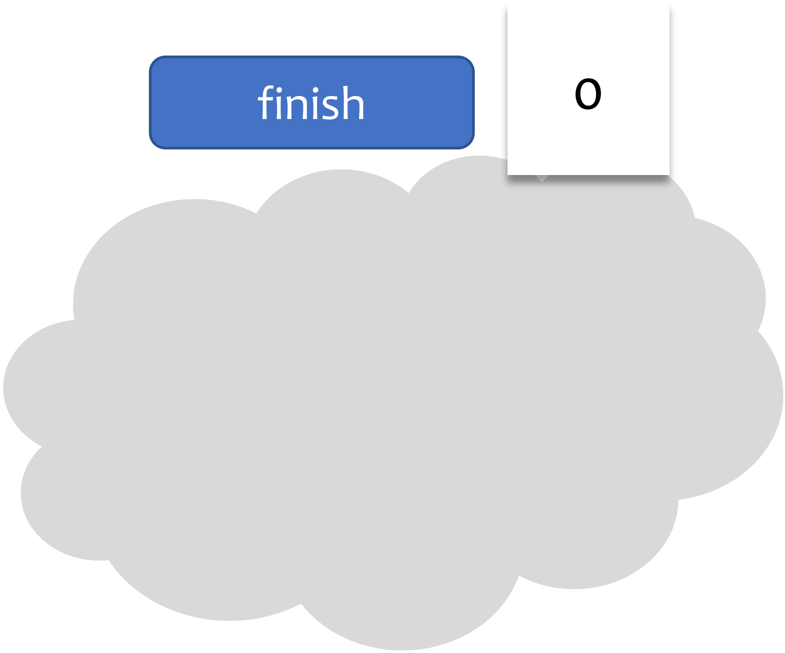
Finish Termination Detection

- Finish tracks the number of active tasks within its scope.



Finish Termination Detection

- Finish tracks the number of active tasks within its scope.



finish

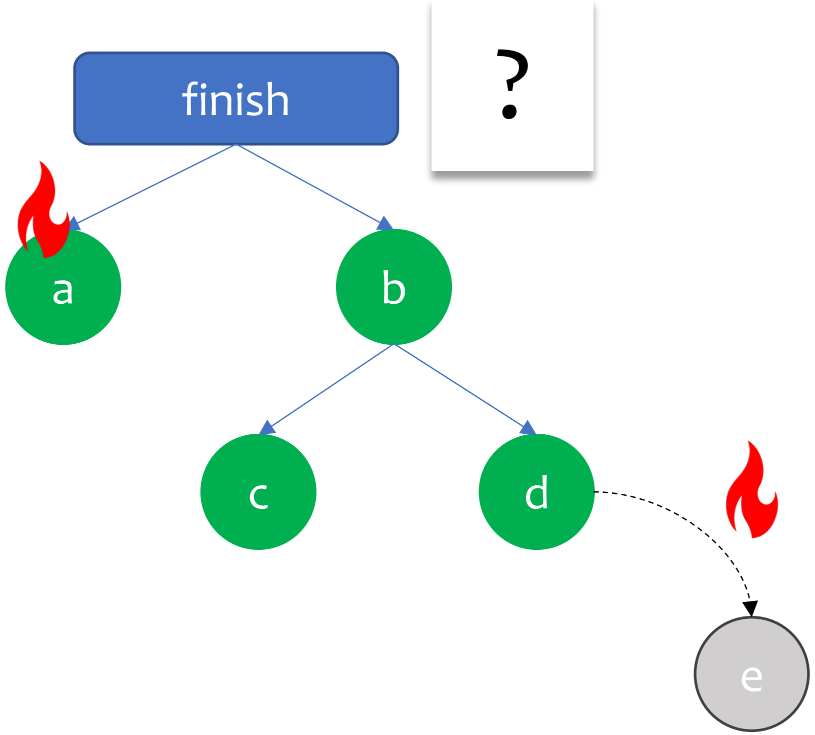
0

Finish Termination Detection

- Finish tracks the number of active tasks within its scope.
- Finish terminates when the number of active tasks reaches zero.

Finish Termination Detection

- Finish tracks the number of active tasks within its scope.
- Finish terminates when the number of active tasks reaches zero.
- Failures complicate the counting process.



Agenda

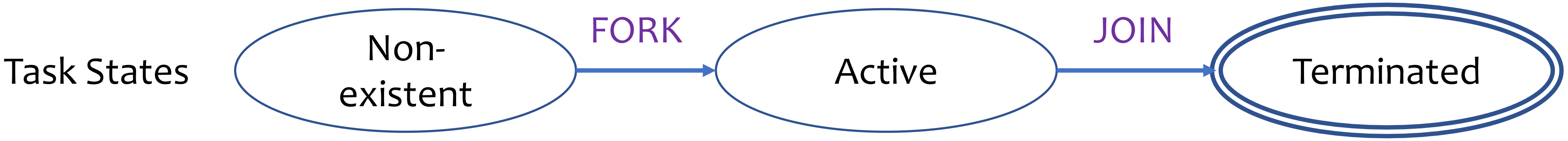
- **Background**
    - The Async-Finish Task Model

- **Async-Finish Termination Detection**
    - The non-resilient protocol
    - The pessimistic protocol
    - The optimistic protocol

- **Performance Evaluation**
    - Microbenchmarks
    - LULESH application

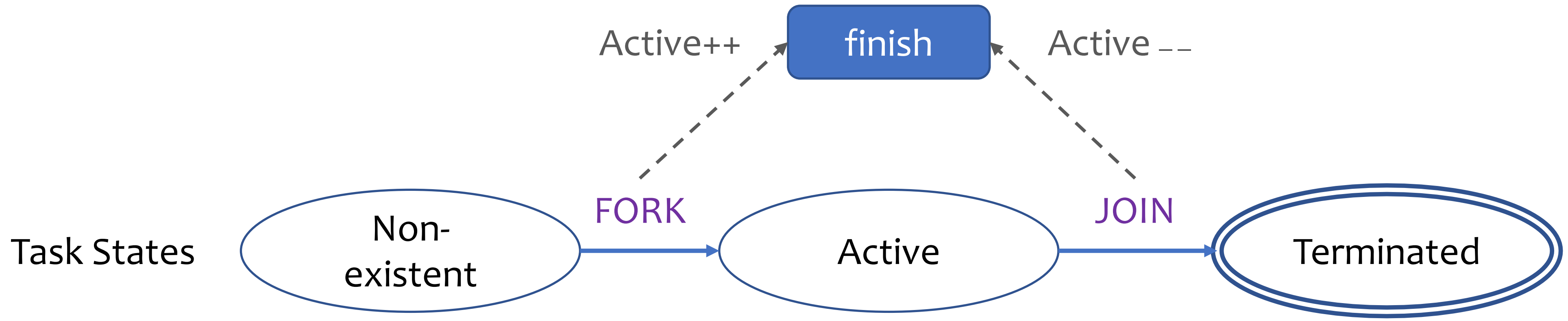
Non-Resilient Finish

- Uses **two TD signals** per task
  - FORK
  - JOIN

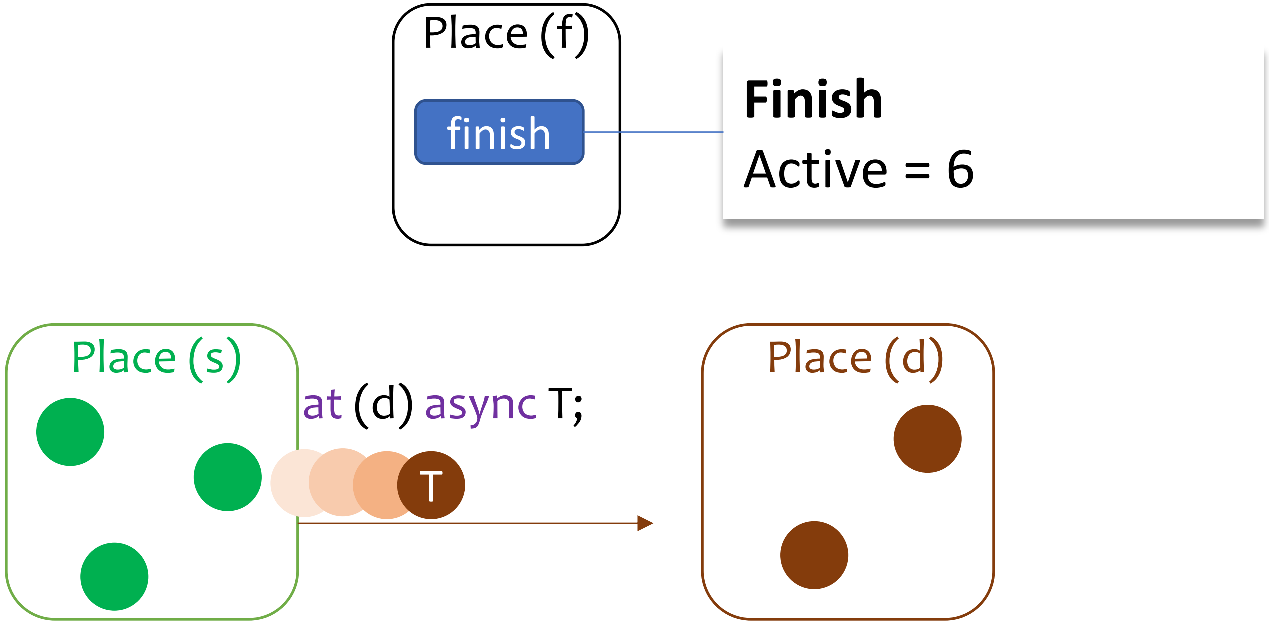


Non-Resilient Finish

- Uses **two TD signals** per task
  - FORK
  - JOIN



Non-Resilient Finish



Place (f)

finish

Finish

Active = 6

Place (s)

at (d) async T;

T

Place (d)

1. **Finish.fork (s, d)**
2. Send (T)

3. Recv (T)
4. Exec (T)
5. **Finish.join (s, d)**

Non-Resilient Finish

- Uses **two TD signals** per task
  - FORK
  - JOIN

- Message-Optimal TD:
  - A correct non-resilient finish requires **one TD message** per task *(see proof in Section 4).*

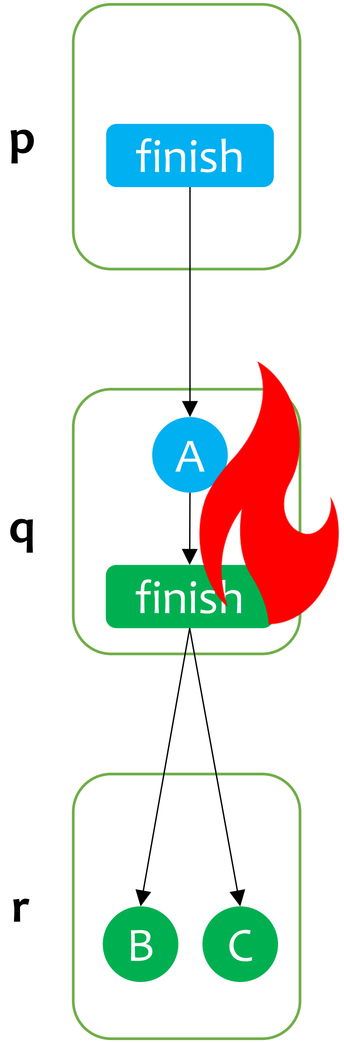
Resilient Finish

Loss of Finish

Loss of Tasks

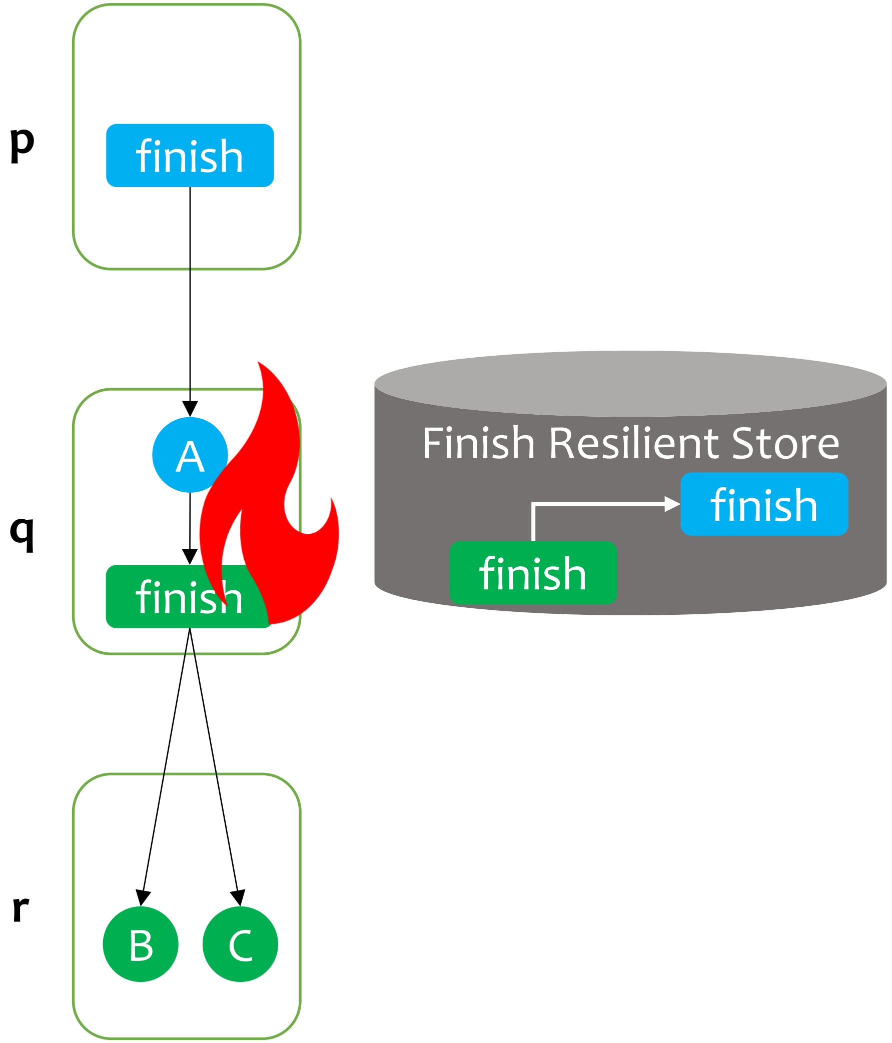
Loss of Finish

- Two problems arise:
  1. Loss of TD metadata.
  2. Emergence of orphan tasks.



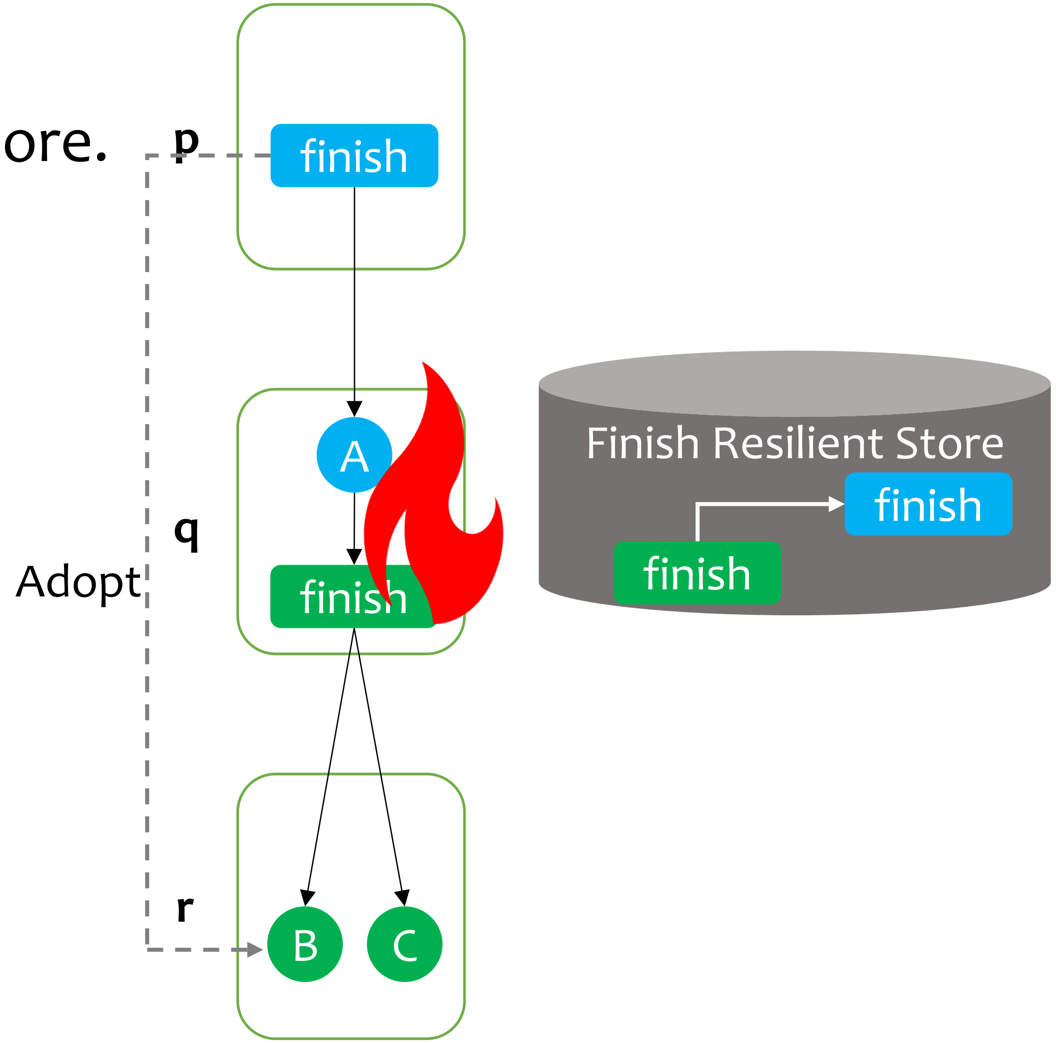
Loss of Finish

- Solutions:
  1. Store the finish objects in a resilient store.



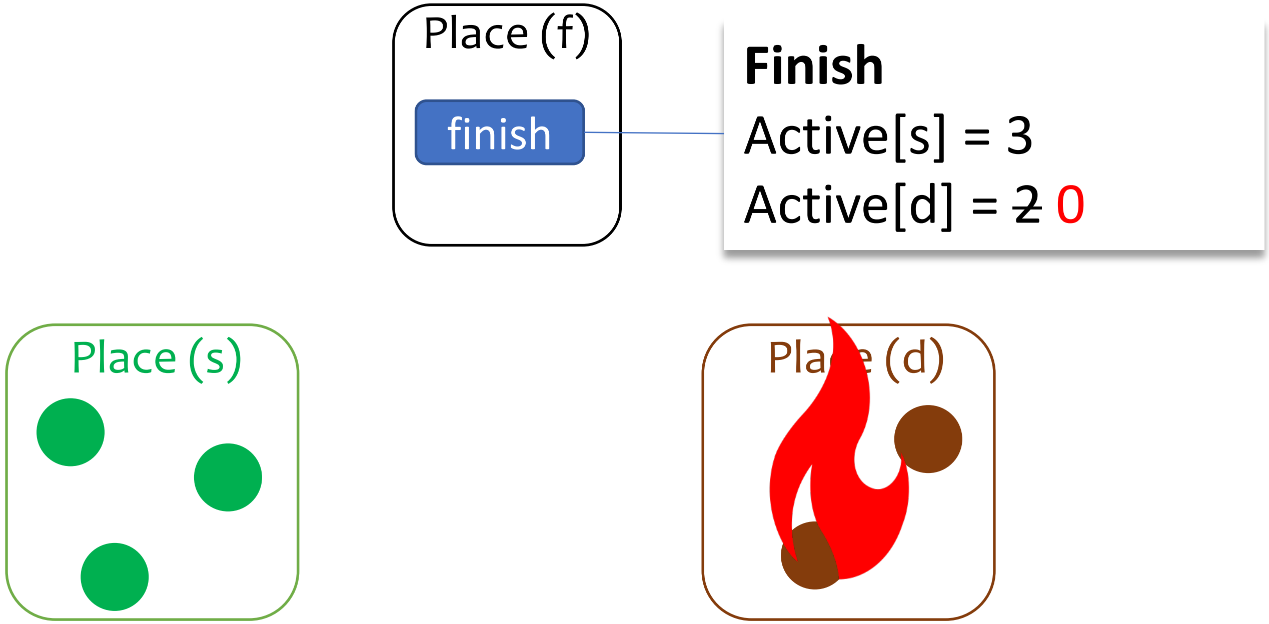
Loss of Finish

- Solutions:
  1. Store the finish objects in a resilient store.
  2. Adoption of orphan tasks.



Loss of Tasks

- Finish must exclude the lost tasks from its count.



Place (f)

finish

Finish

Active[s] = 3

Active[d] = ~~2~~ 0

Place (s)

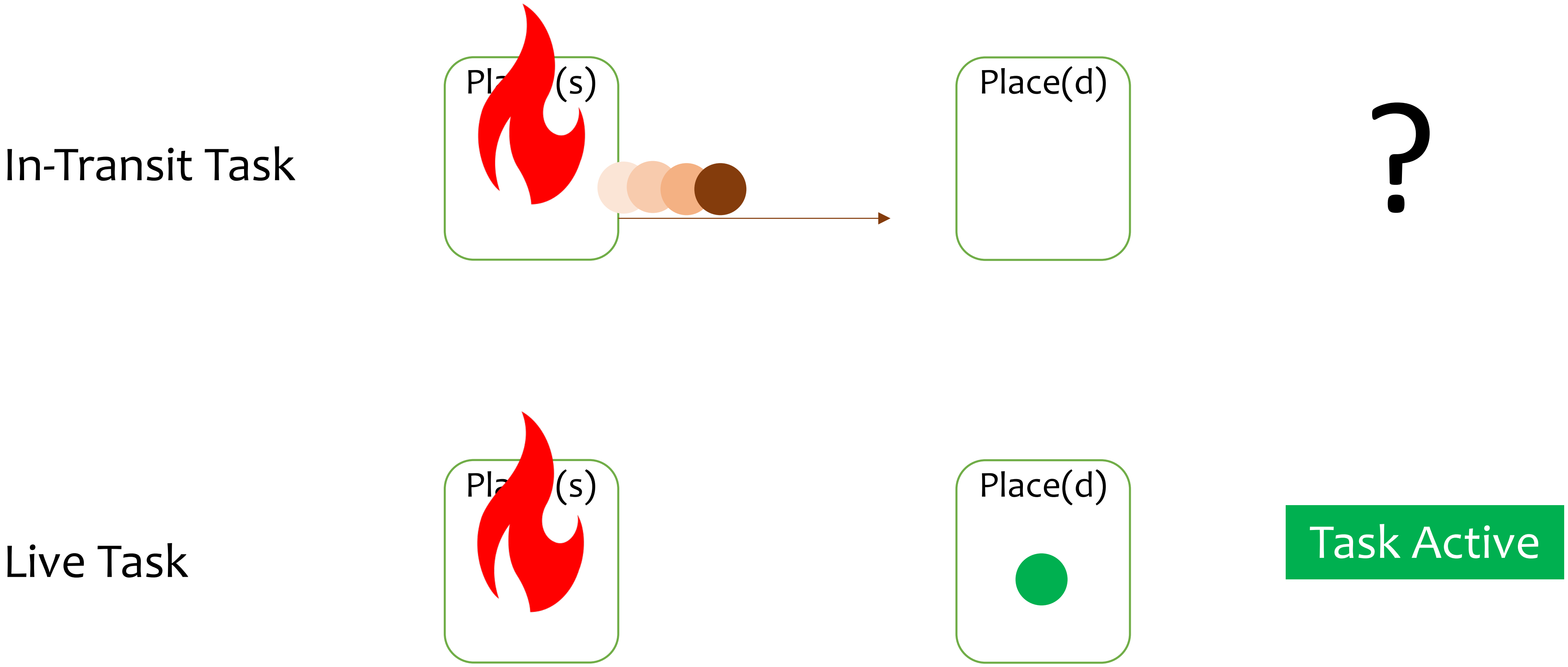
Place (d)

Loss of Tasks

- In-transit and live tasks have different conditions under failure.

Loss of Tasks

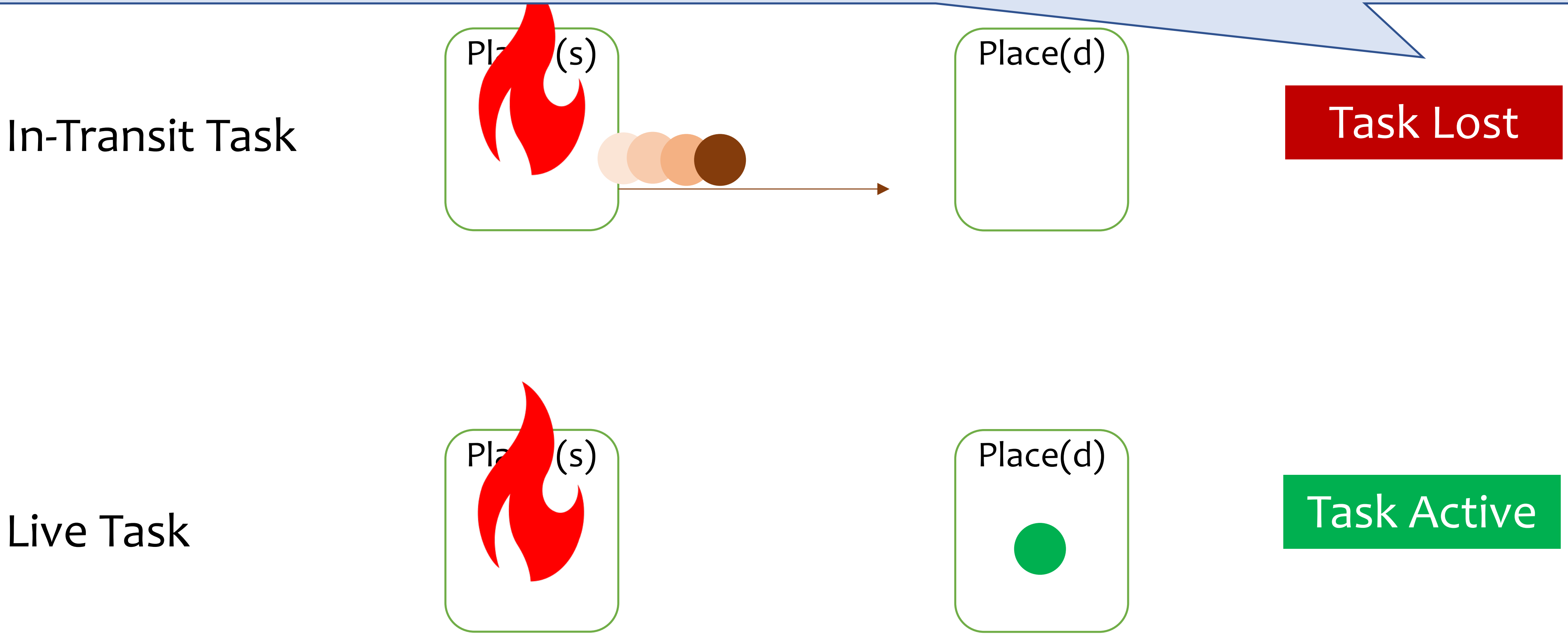
- In-transit and live tasks have different conditions under failure.
- Failure of the source:



Loss of Tasks

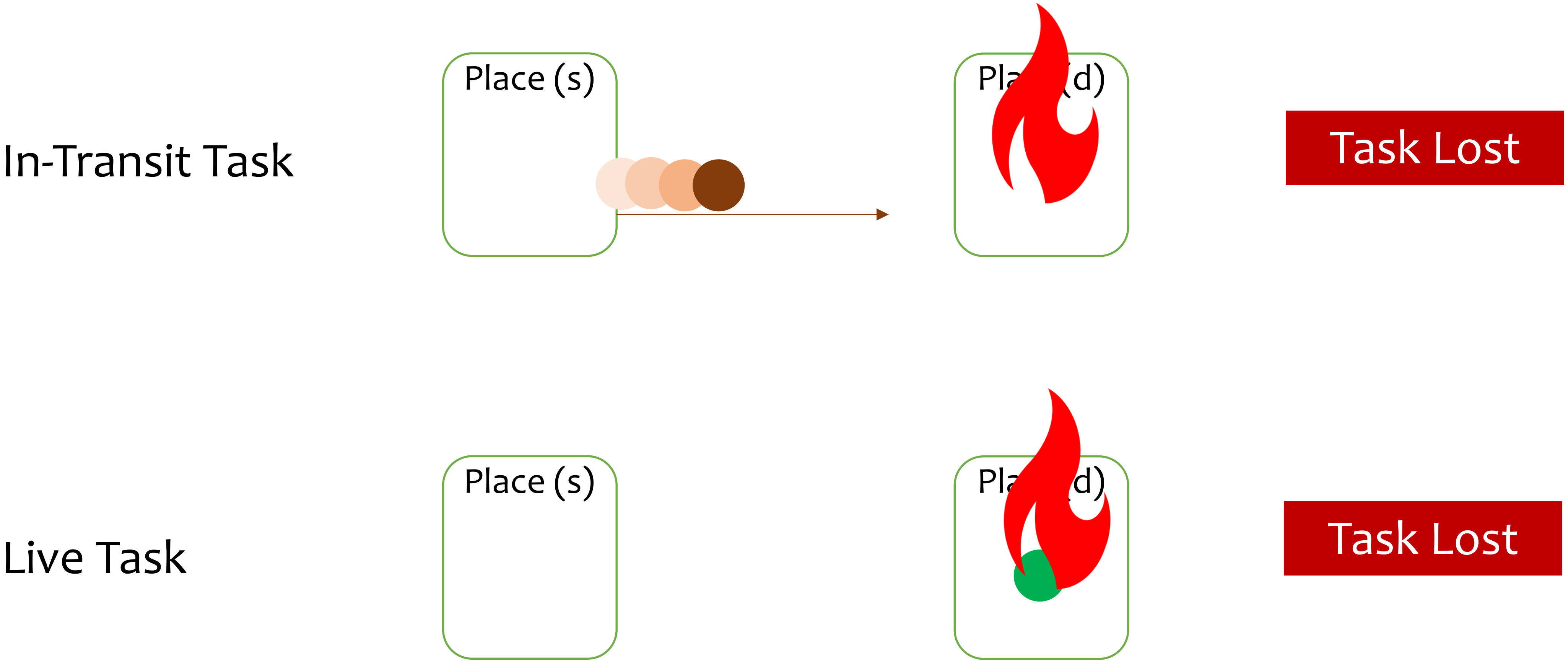
To avoid indefinite waiting

- Consider in-transit tasks from a dead source lost
- A destination must not execute a task whose source is dead



Loss of Tasks

- In-transit and live tasks have different conditions under failure.
- Failure of the destination:



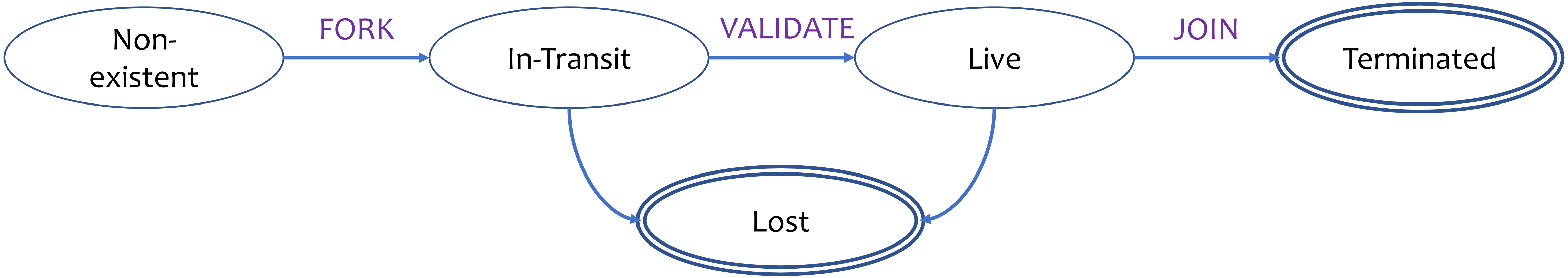
Loss of Tasks

- For recovery, it is important to differentiate between in-transit tasks and live tasks.
    - Finish excludes **all tasks** (in-transit or live) targeted to a dead place.
    - Finish excludes **only in-transit tasks** originated from a dead place.

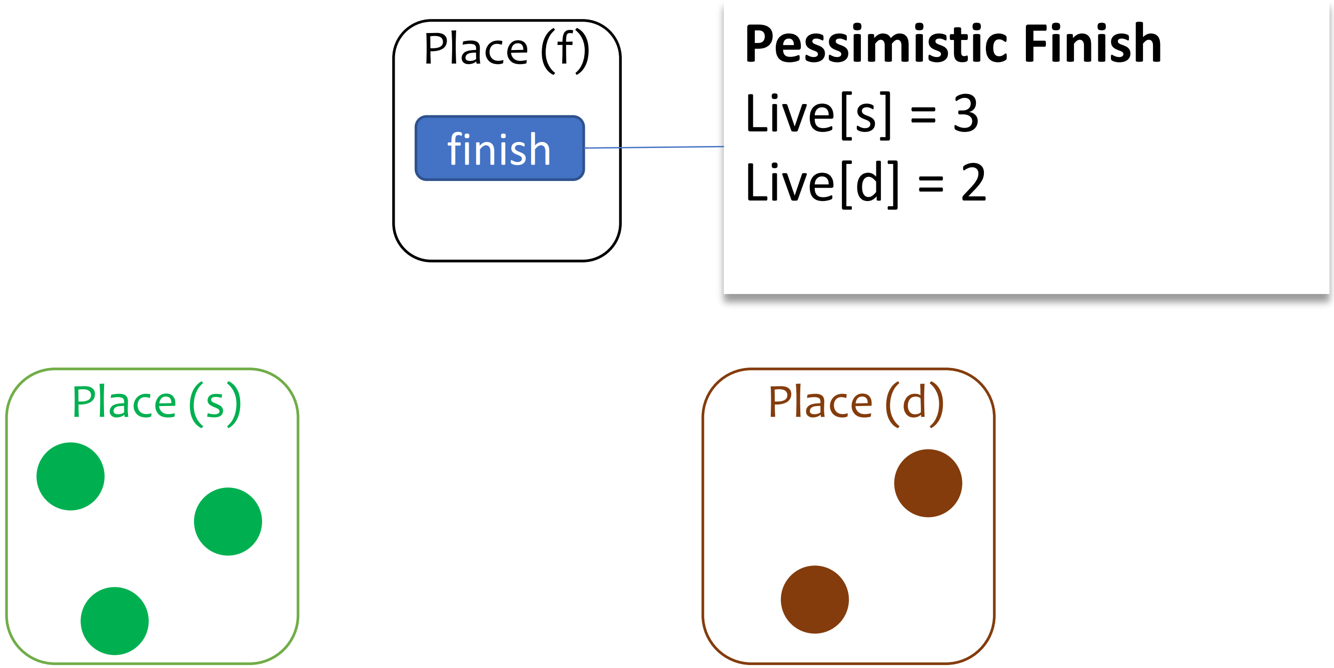
- Message-Optimal TD:
    - A correct resilient finish requires **two TD messages** per task (see proof in Section 4).
        - Message for the FORK signal
        - Message for the JOIN signal

Pessimistic Finish

- Uses **three TD messages** per task (not message-optimal)
  - FORK
  - VALIDATE
  - JOIN



Pessimistic Finish



Pessimistic Finish

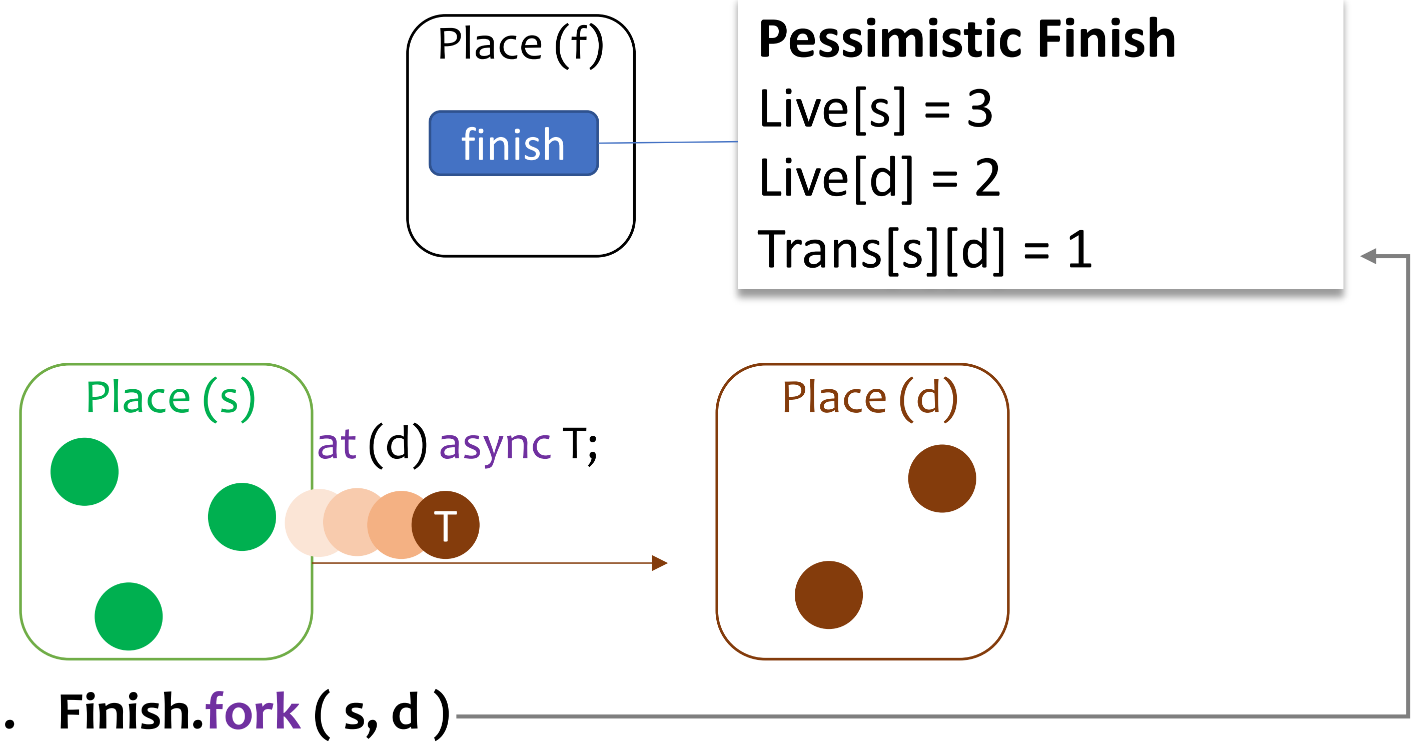
Live[s] = 3

Live[d] = 2

1. **Finish.fork (s, d)**
2. Send (T)

3. Recv (T)
4. **Finish.validate (s, d)**
5. Exec (T)
6. **Finish.join (s, d)**

Pessimistic Finish

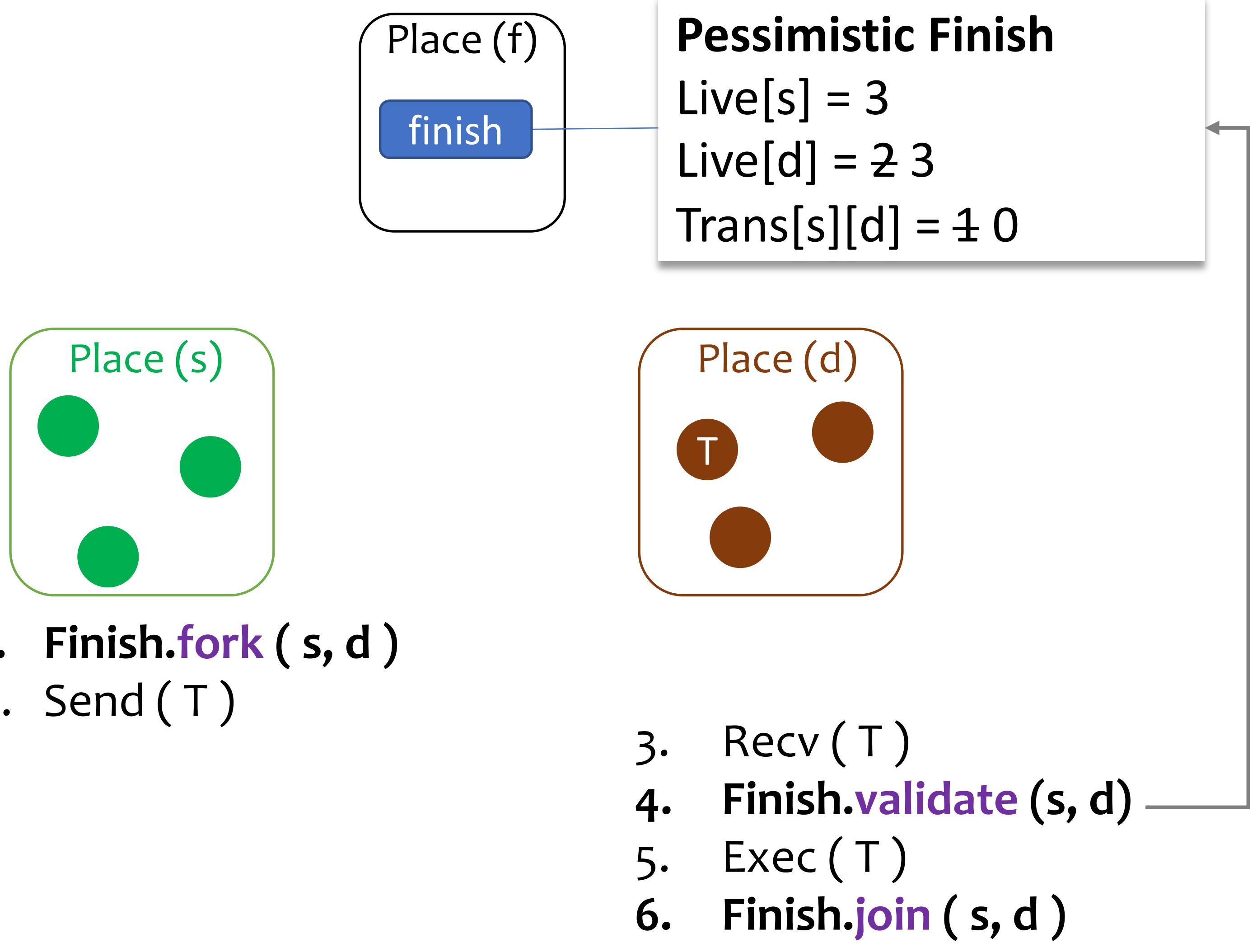


Pessimistic Finish
Live[s] = 3
Live[d] = 2
Trans[s][d] = 1

1. **Finish.fork (s, d)**
2. Send (T)

3. Recv (T)
4. **Finish.validate (s, d)**
5. Exec (T)
6. **Finish.join (s, d)**

Pessimistic Finish

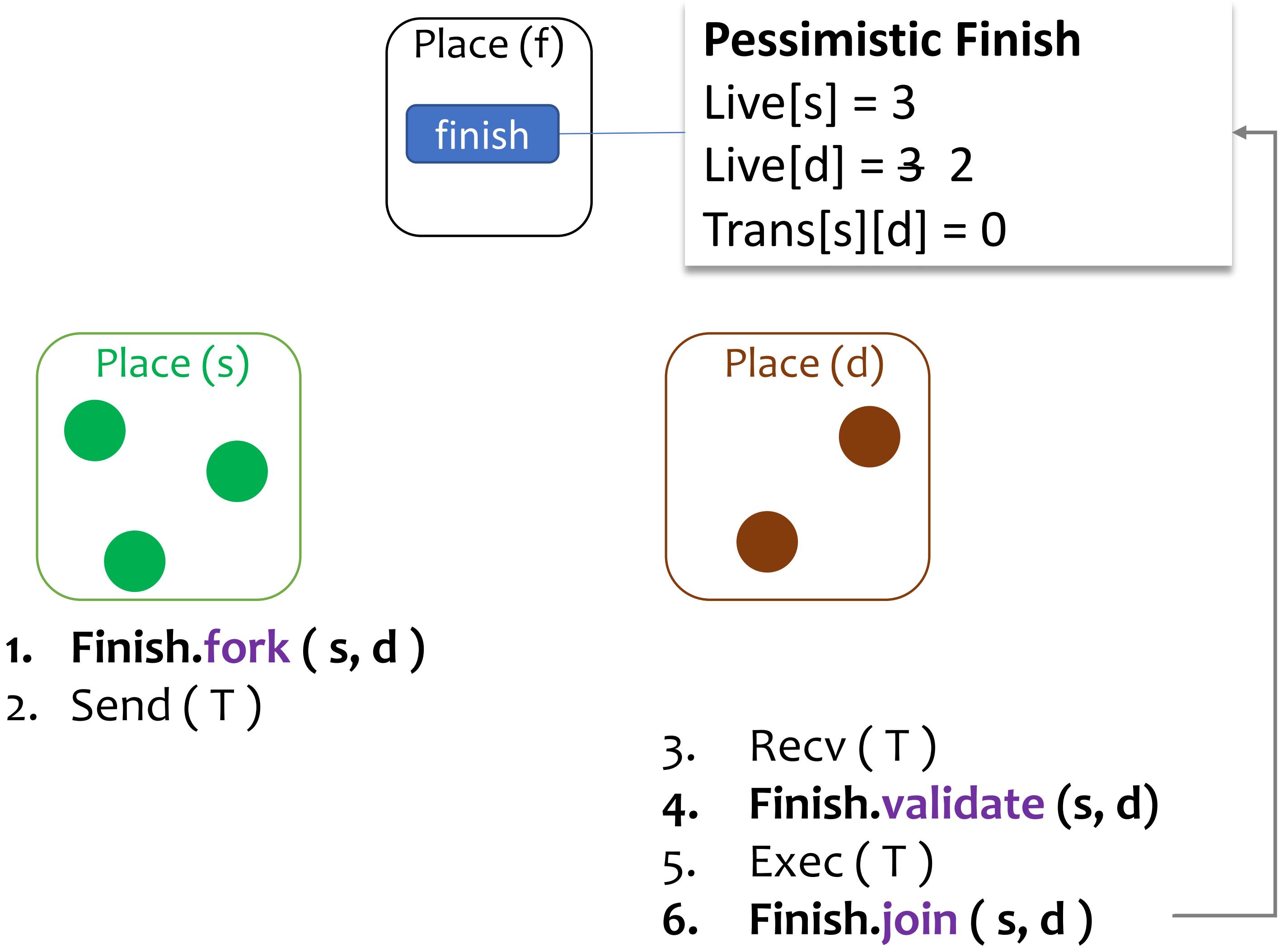


Pessimistic Finish
Live[s] = 3
Live[d] = ~~2~~ 3
Trans[s][d] = ~~1~~ 0

1. **Finish.fork (s, d)**
2. Send (T)

3. Recv (T)
4. **Finish.validate (s, d)**
5. Exec (T)
6. **Finish.join (s, d)**

Pessimistic Finish



Pessimistic Finish

Live[s] = 3

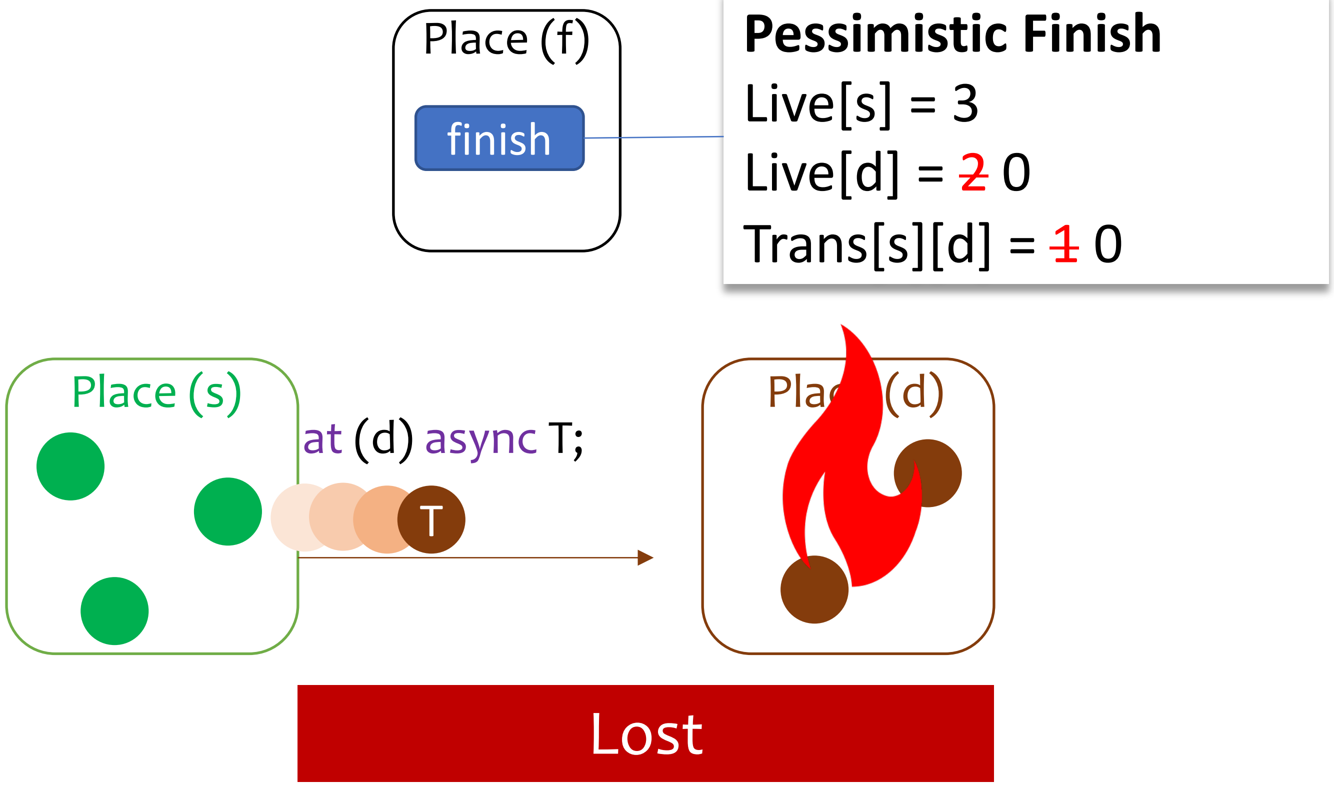
Live[d] = ~~3~~ 2

Trans[s][d] = 0

1. **Finish.fork (s, d)**
2. Send (T)

3. Recv (T)
4. **Finish.validate (s, d)**
5. Exec (T)
6. **Finish.join (s, d)**

Pessimistic Finish

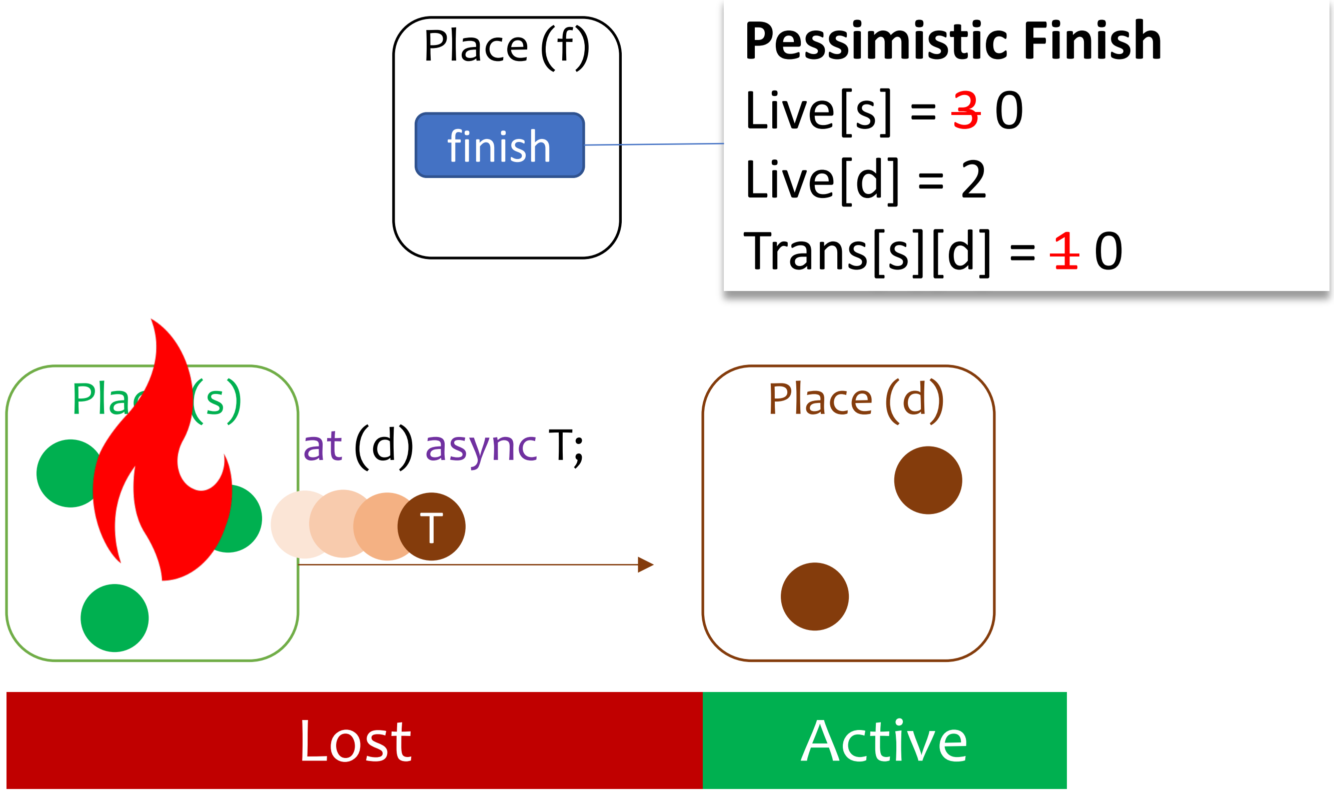


at (d) async T;

Pessimistic Finish
Live[s] = 3
Live[d] = ~~2~~ 0
Trans[s][d] = ~~1~~ 0

Lost

Pessimistic Finish



Pessimistic Finish

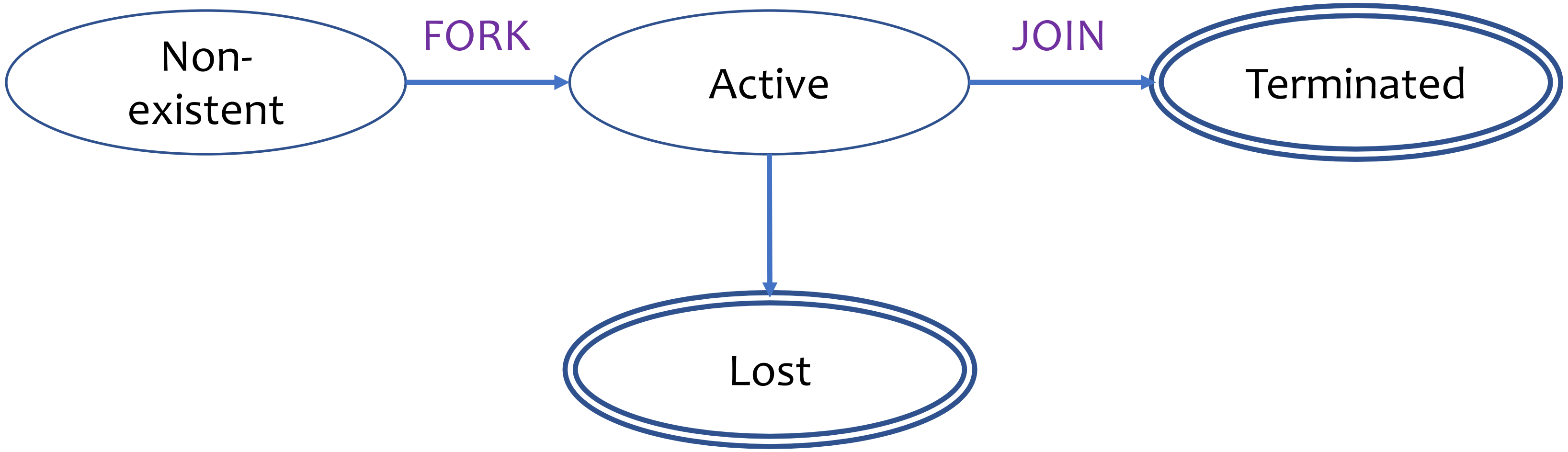
Live[s] = ~~3~~ 0

Live[d] = 2

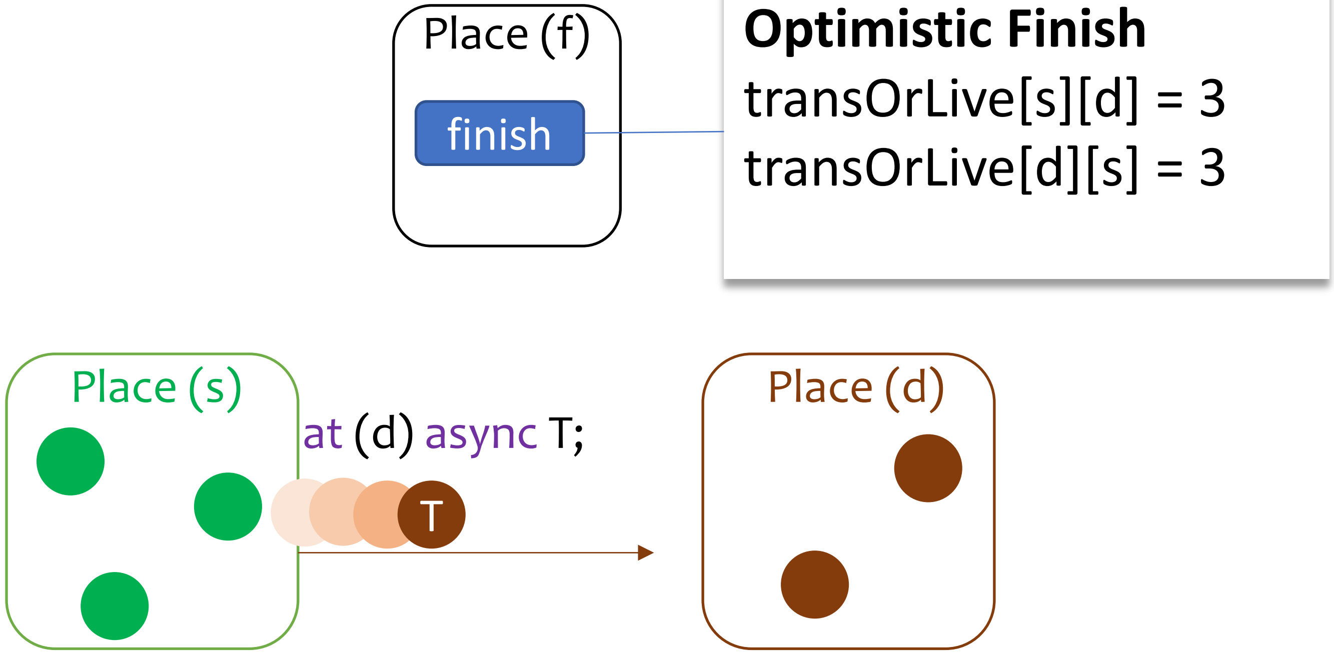
Trans[s][d] = ~~1~~ 0

Optimistic Finish

- Uses **two TD messages** per task (message-optimal)
  - FORK
  - JOIN



Optimistic Finish

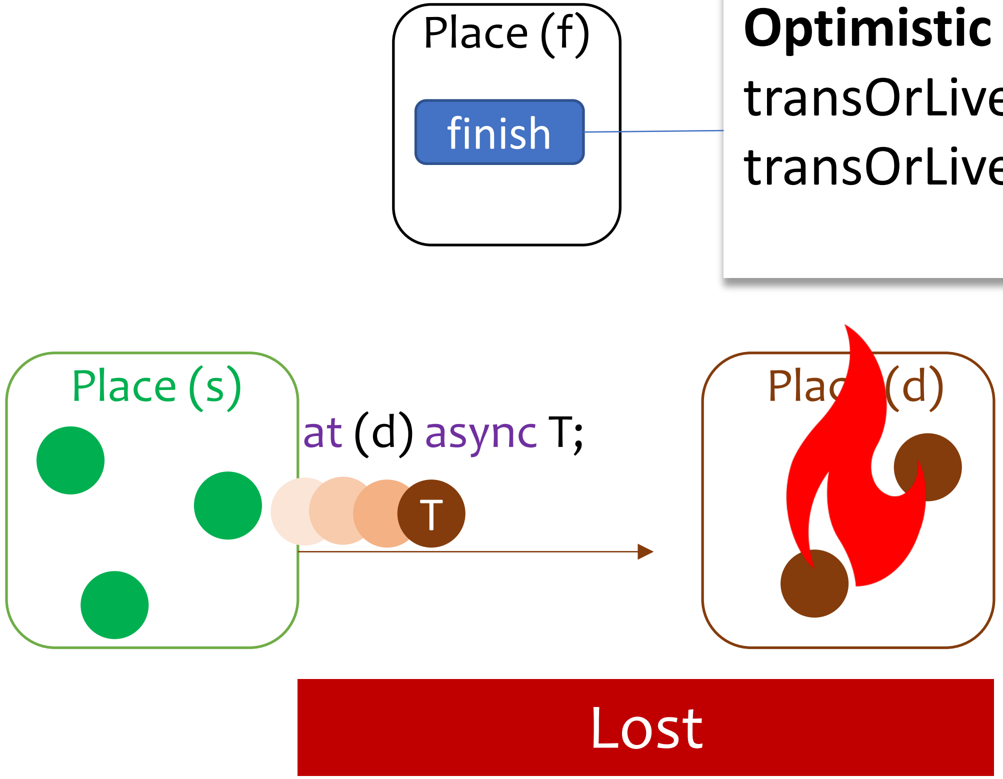


Optimistic Finish
transOrLive[s][d] = 3
transOrLive[d][s] = 3

1. **Finish.fork (s, d)**
2. Send (T)

3. Recv (T)
4. ~~**Finish.validate (s, d)**~~
5. Exec (T)
6. **Finish.join (s, d)**

Optimistic Finish

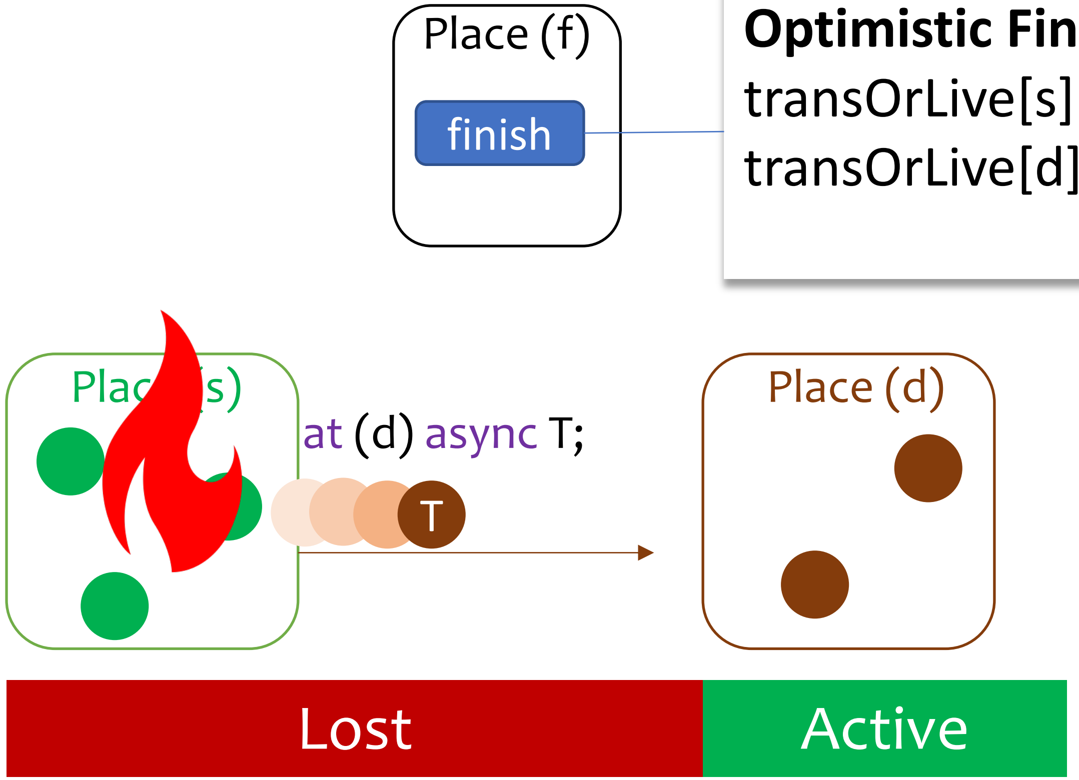


Optimistic Finish

transOrLive[s][d] = ~~3~~ 0

transOrLive[d][s] = 3

Optimistic Finish



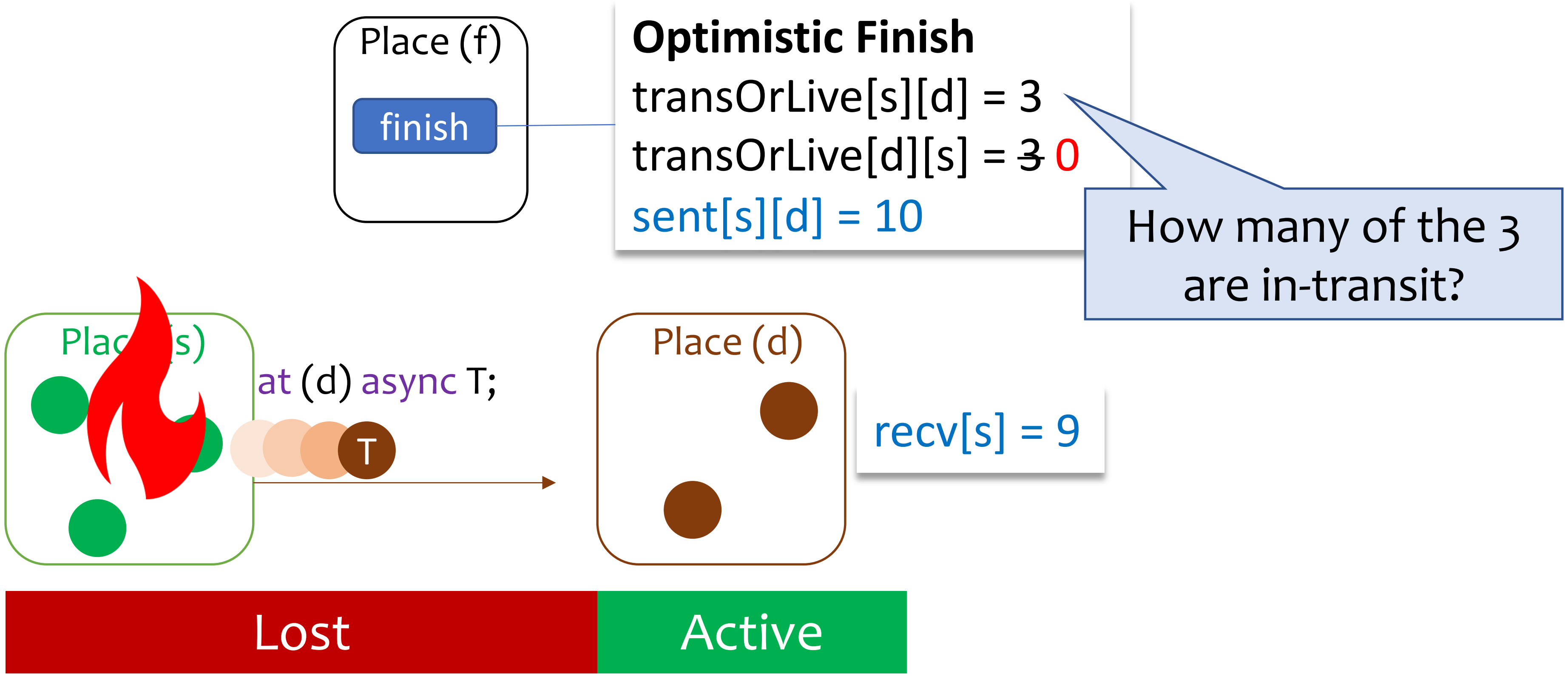
Optimistic Finish

transOrLive[s][d] = 3

transOrLive[d][s] = ~~3~~ 0

How many of the 3 are in-transit?

Optimistic Finish

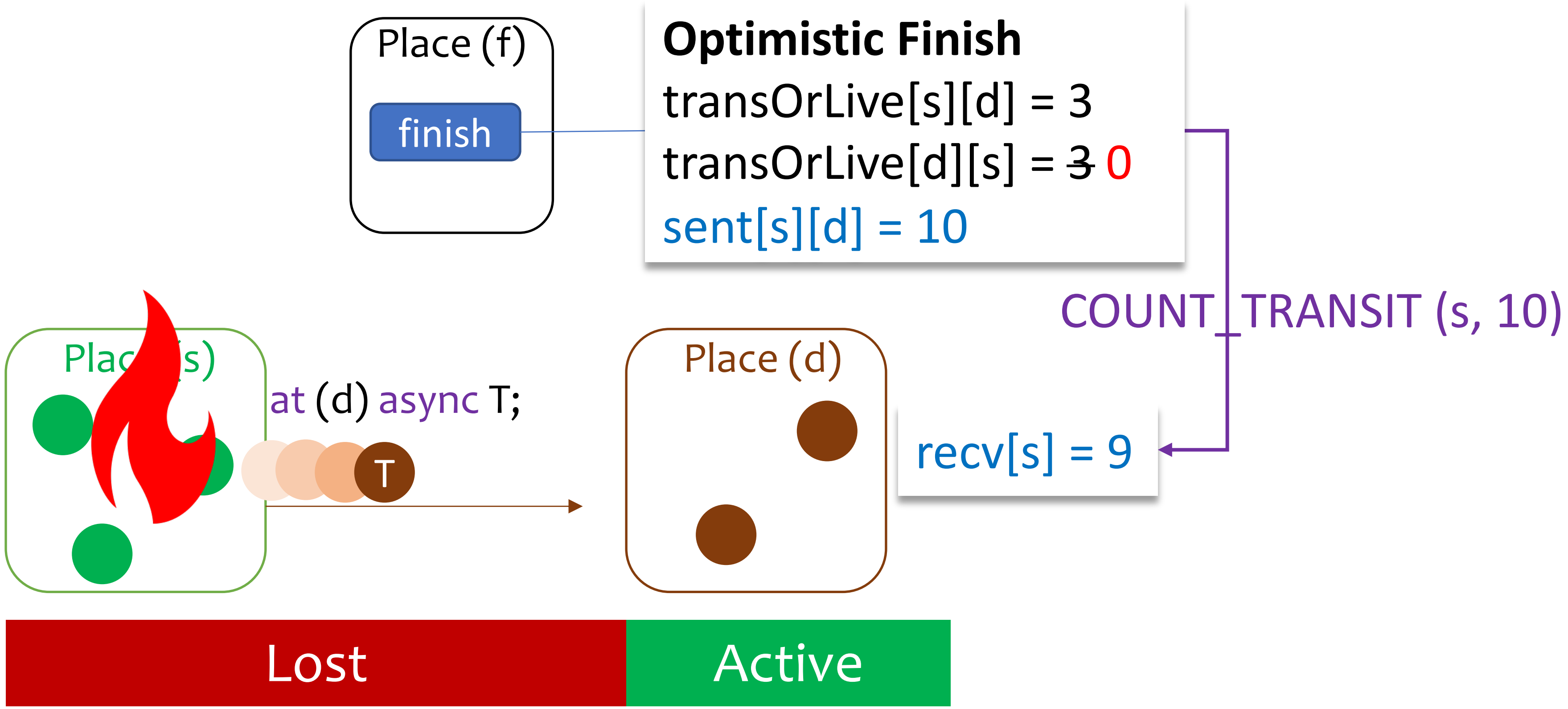


Optimistic Finish
transOrLive[s][d] = 3
transOrLive[d][s] = ~~3~~ 0
sent[s][d] = 10

recv[s] = 9

How many of the 3 are in-transit?

Optimistic Finish



Optimistic Finish

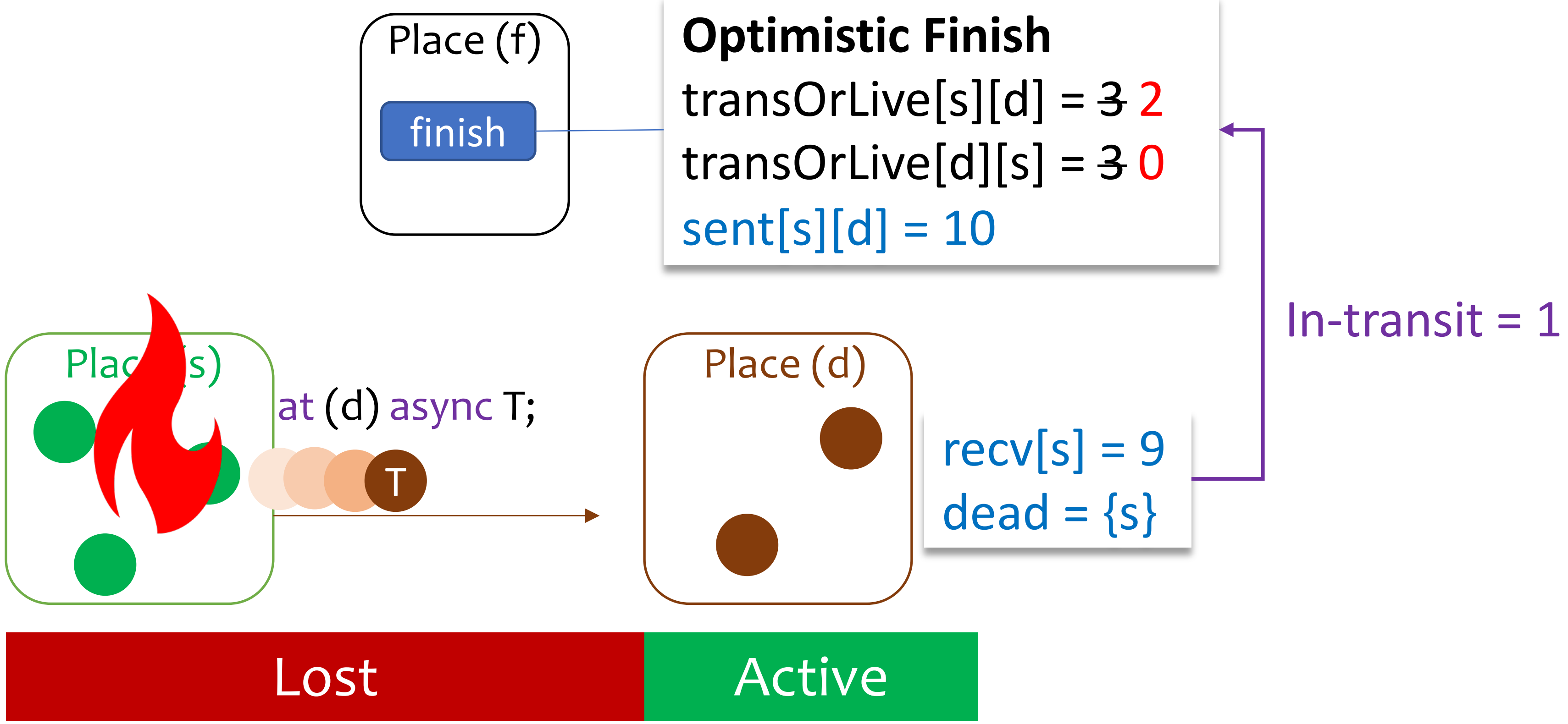
transOrLive[s][d] = 3

transOrLive[d][s] = ~~3~~ 0

sent[s][d] = 10

recv[s] = 9

Optimistic Finish



Optimistic Finish

transOrLive[s][d] = ~~3~~ 2

transOrLive[d][s] = ~~3~~ 0

sent[s][d] = 10

recv[s] = 9

dead = {s}

In-transit = 1

Resilient Termination Detection Signals

Pessimistic Finish

- Task signals
  - FORK
  - VALIDATE
  - JOIN
- Finish signals:
  - PUBLISH
  - ADD_CHILD
  - RELEASE
- Recovery signals:
  - None

Optimistic Finish

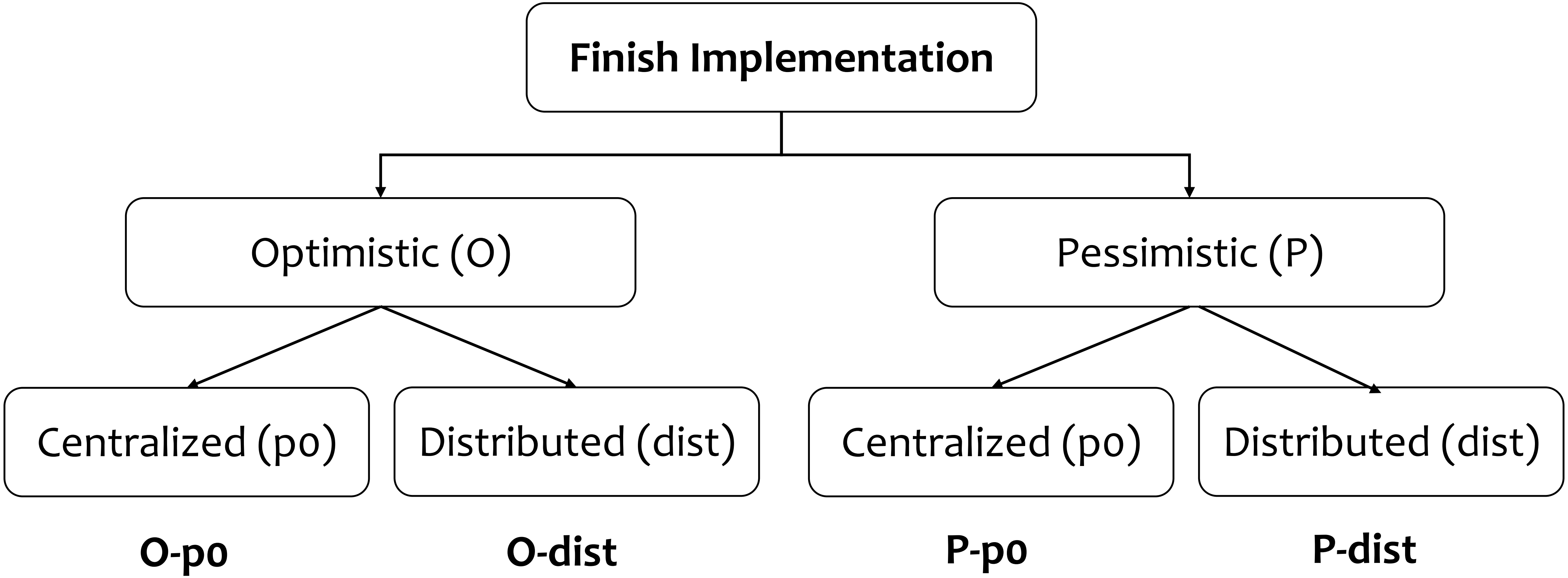
- Task signals:
  - FORK
  - JOIN
- Finish signals:
  - PUBLISH
  - RELEASE
- Recovery signals:
  - COUNT_TRANSIT
  - FIND_CHILDREN

Optimistic Finish Correctness

- We verified the correctness of our protocol using TLA+ Model Checker.
- Specification:
  - https://github.com/shamouda/x10-formal-spec
- See section 8.3 for the details.

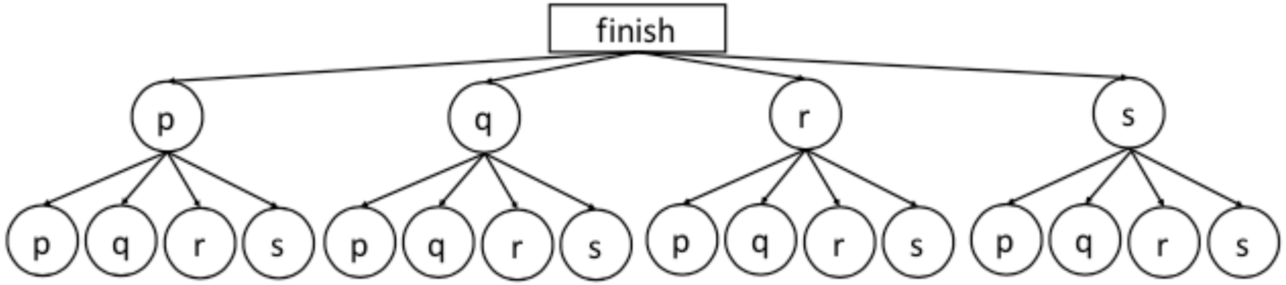
Performance Evaluation

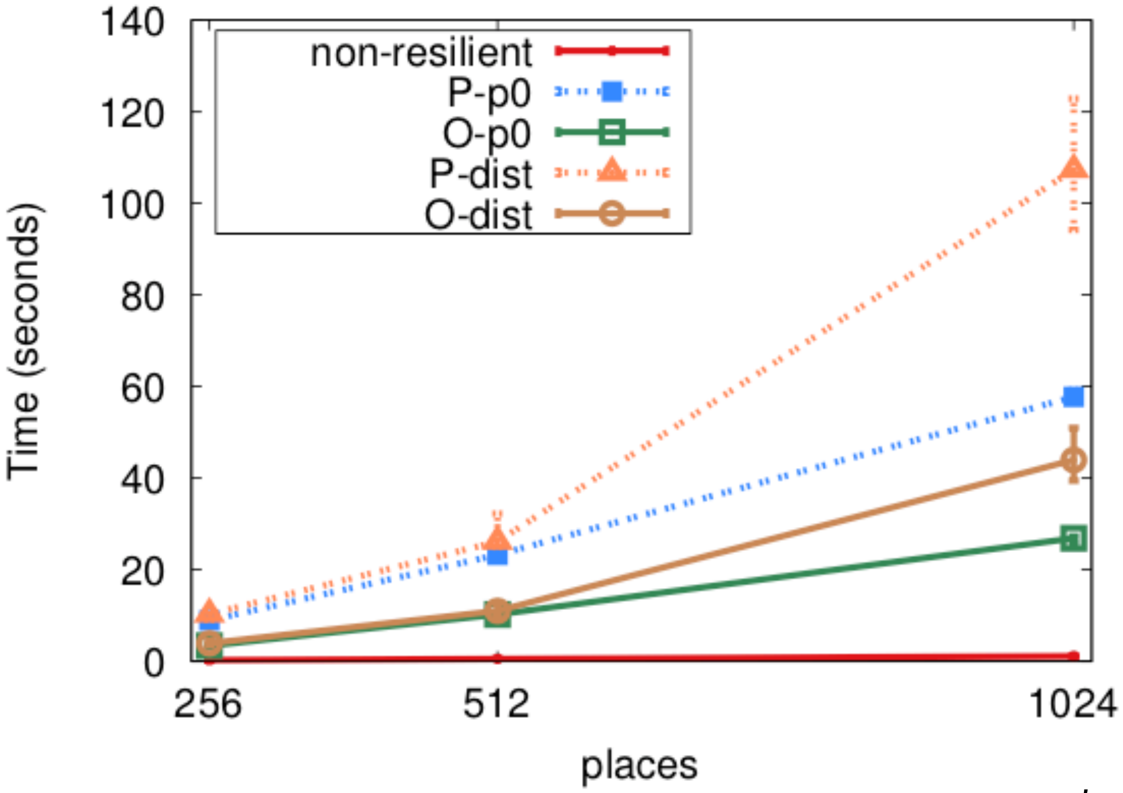
Finish Implementations



Microbenchmarks

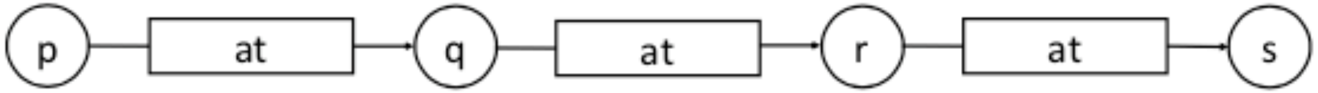
- Fan-Out Fan-Out (All-to-all)
  - At 1024 places:
    - Tasks/Finish: 1024^2
    - Improvement centralized: **53%**
    - Improvement distributed: **59%**

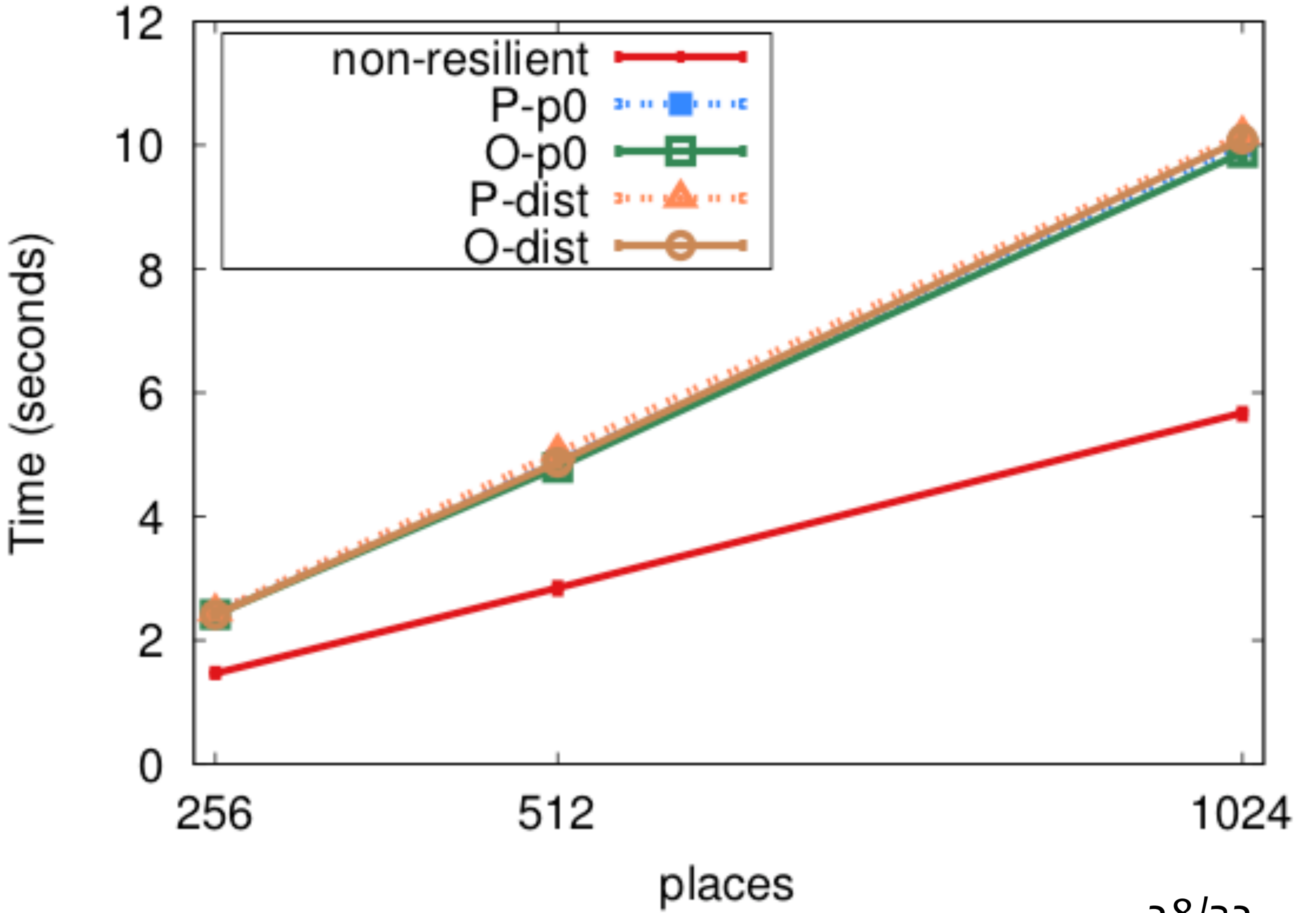




Microbenchmarks

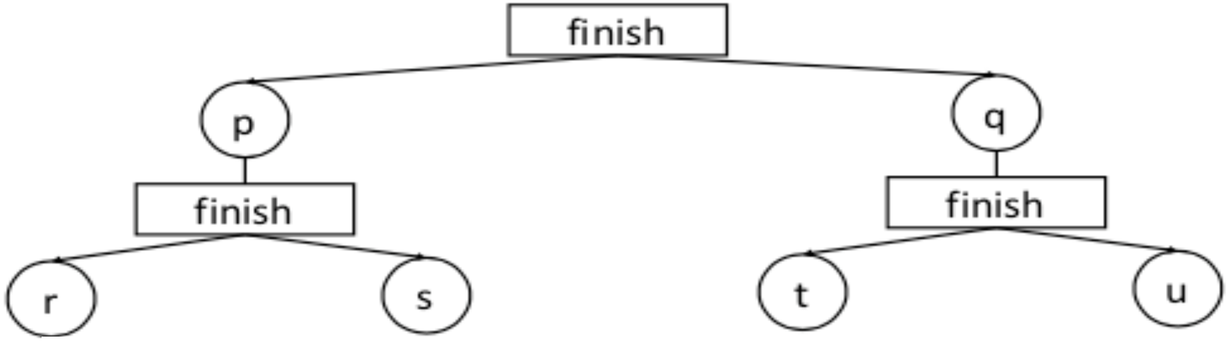
- Synchronous Ring
  - At 1024 places:
    - Tasks/Finish: **1**
    - Improvement Centralized: **1%**
    - Improvement Distributed: **0%**

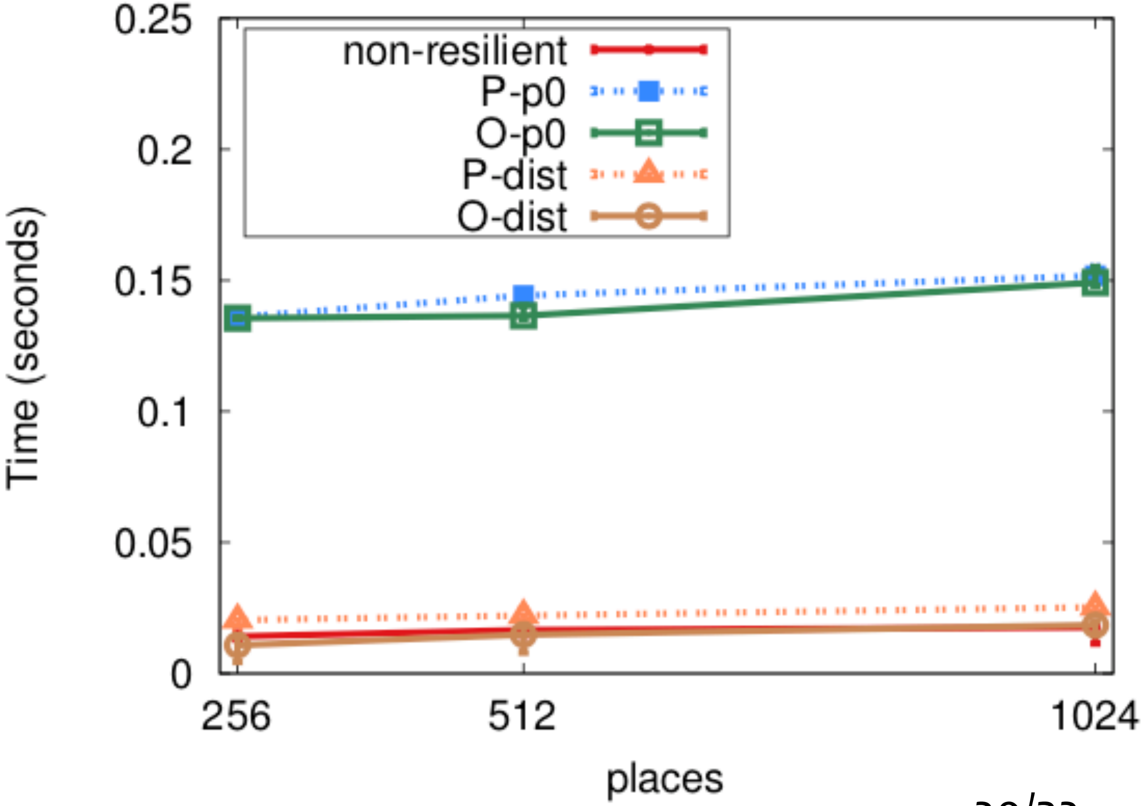




Microbenchmarks

- Binary Tree Fan-Out
  - At 1024 places:
    - Tasks/Finish: **2**
    - Improvement centralized: **2%**
    - Improvement distributed: **27%**

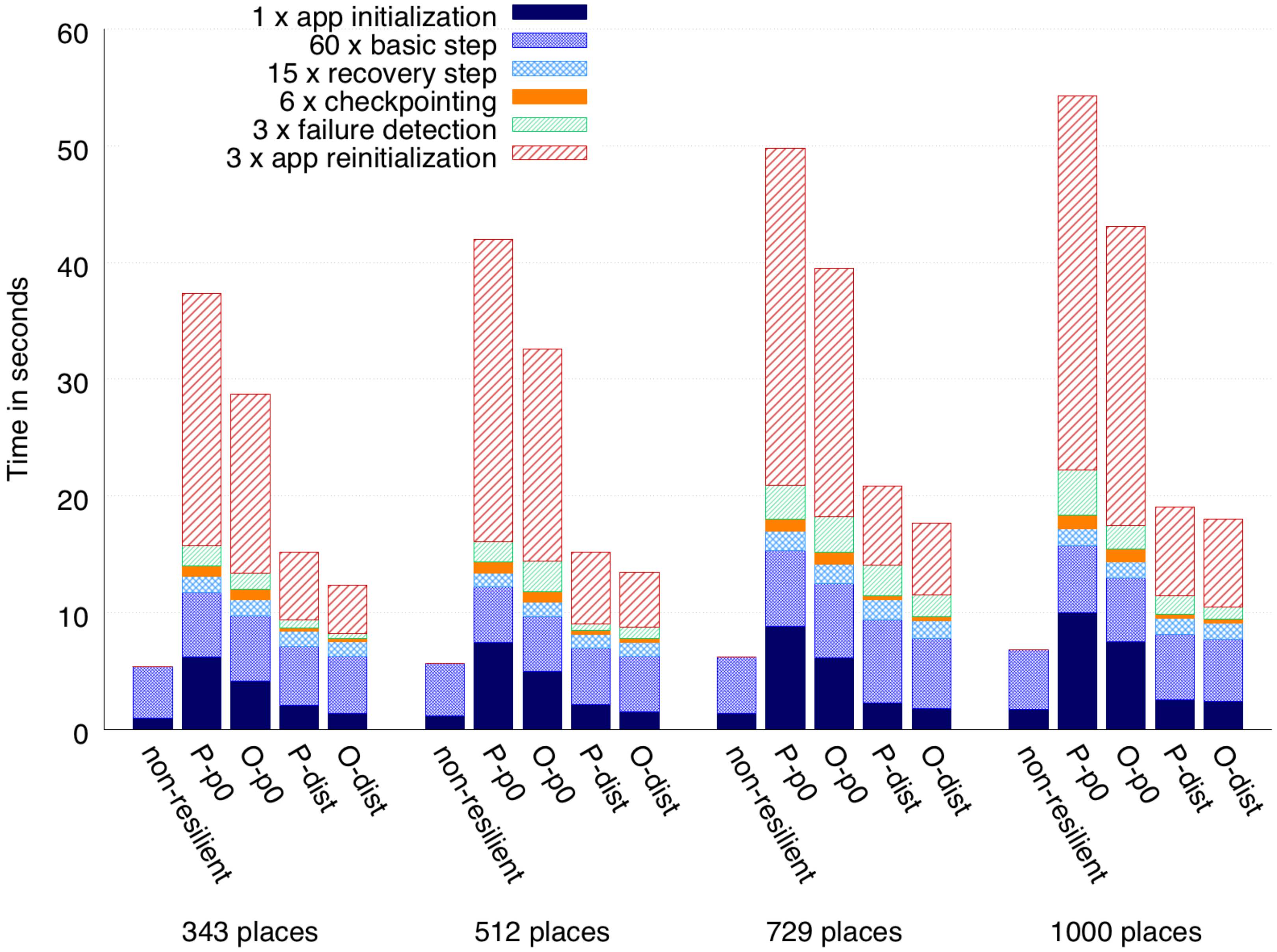




LULESH

- A shock hydrodynamics proxy application.
  - Iterative
  - Stencil-based
- X10's implementation:
  - In-memory checkpointing
  - Communication intensive initialization module
    - Called at the beginning of execution.
    - Called at failure recovery time.
- Failure simulation:
  - Execute 60 iterations
  - Checkpoint every 10 iteration
  - Kill 3 places at iterations: 15, 35, 55

LULESH


1 x app initialization
60 x basic step
15 x recovery step
6 x checkpointing
3 x failure detection
3 x app reinitialization
Time in seconds
0
10
20
30
40
50
60
non-resilient
P-p0
O-p0
P-dist
O-dist
343 places
512 places
729 places
1000 places

Summary

- We presented 'Optimistic Finish' -- a message-optimal resilient termination detection protocol for the async-finish model.
    - The effect of the optimistic protocol is more evident as the number of remote tasks increases.

- **Takeaway message:** Simple reductions in runtime tracking messages can result in significant performance improvements.

- It is open-source:
    - Source code: https://github.com/shamouda/x10/tree/optimistic
    - TLA+ Specification: https://github.com/shamouda/x10-formal-spec

Thank you!