ALGORITHMS: RECOVERABLE MUTEX AND CONSENSUS

Presenter: Wojciech Golab
wgolab@uwaterloo.ca

PODC 2019

Toronto

August 2nd, 2019

UNIVERSITY OF WATERLOO

OUTLINE

- **Background**
- **Recoverable Mutex**
- **Recoverable Consensus**

BACKGROUND

PROCESS VS. THREAD

Theory:

process = thread

Practice:

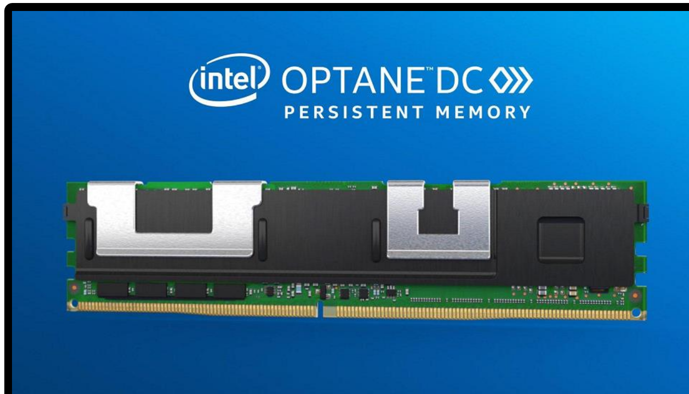
process = collection of parallel threads

This talk: theory

MEMORY HIERARCHY

CPU registers
L1/L2/L3 cache

DRAM

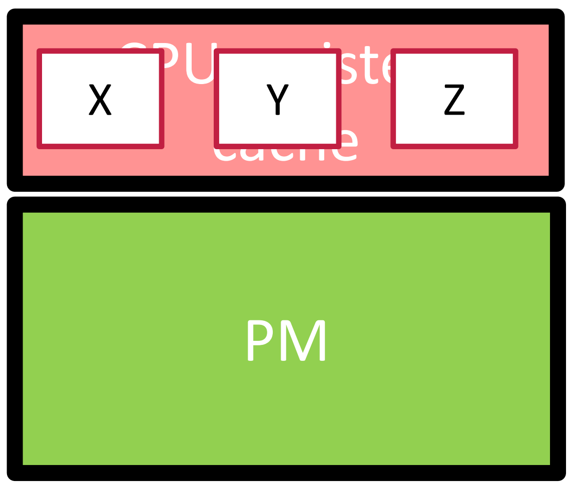


secondary storage

WHAT HAPPENS DURING A FAILURE?

Case 1: system-wide failure (reboot)

- power outage
- kernel panic

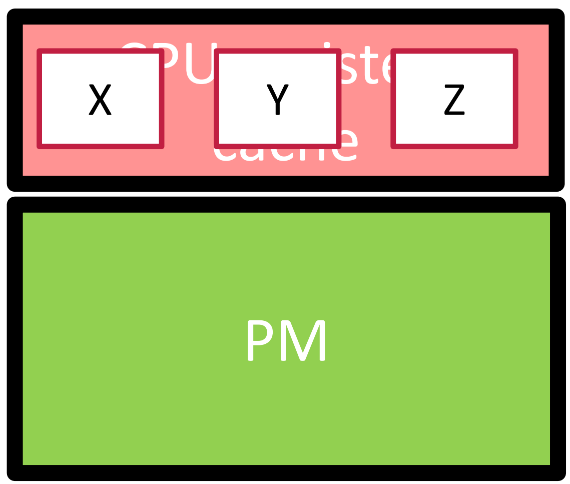


1. write X
2. write Y
3. write Z
4. crash

WHAT HAPPENS DURING A FAILURE?

Case 2: individual process failure (no reboot)

- software bug
- uncaught exception
- deadlock breakup



1. write X
2. write Y
3. write Z
4. crash

…

WHAT HAPPENS DURING A FAILURE?

Case 2: individual process failure (no reboot)

Question: Did we lose anything important?

Answer:

Yes, the program counter, stack pointer, and the values of certain program variables.

WHAT HAPPENS DURING A FAILURE?

Example:

```
if T.TestAndSet() then
    // loser
else
    // winner
end if
```

WHAT HAPPENS DURING A FAILURE?

Example:

```
CPU register := T.TestAndSet()
if CPU register = 1 then
    // loser
else
    // winner
end if
```

INSIGHT

In both failure modes, we should be concerned with the potential loss of the response to a Write or Read-Modify-Write operation on a shared variable.

Exception: multi-reader single-writer registers.

ASSUMPTIONS

1. Asynchronous shared memory.
2. Crash-recovery failures (system-wide or independent).
3. Max number of processes N known ahead of time.
4. Participation by all processes is not required (e.g., possible that only $k < N$ processes take steps in an execution).
5. Read-Modify-Write primitives return responses in volatile CPU registers.

OBSERVATIONS

1. In an execution involving *N* processes, the maximum number of failures is not bounded by *N*. It is unbounded!

2. In an execution containing infinitely many independent failures, it is possible that some processes fail only a finite number of times.

RECOVERABLE MUTEX

MUTUAL EXCLUSION PROBLEM

loop forever

Non-Critical/remainder Section

Enter

Critical Section (CS)

Exit

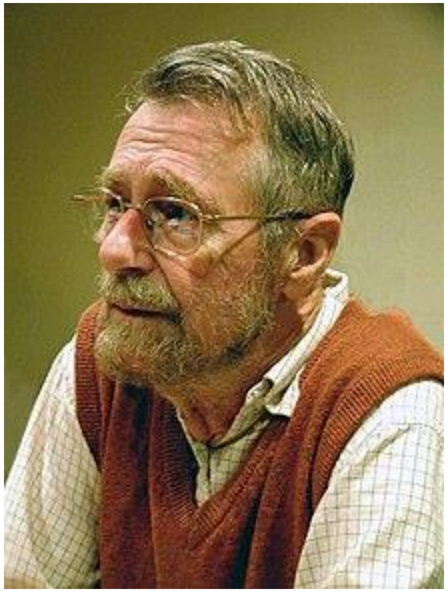


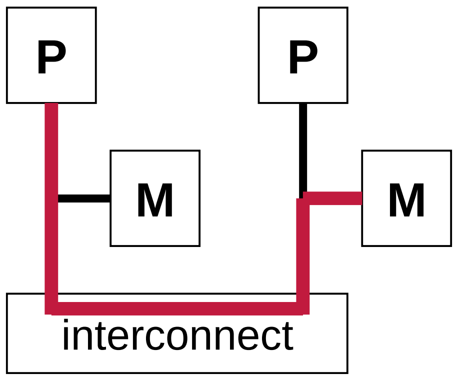
Image source: Wikipedia
https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

Asynchronous, reliable processes.

REMOTE MEMORY REFERENCES (RMR)

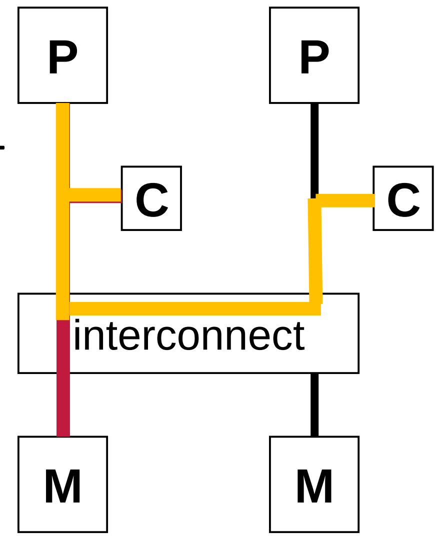
Distributed Shared Memory (DSM)

remote read or write



Cache-Coherent (CC)

read + cache miss



write + invalidation

Legend:

P processor

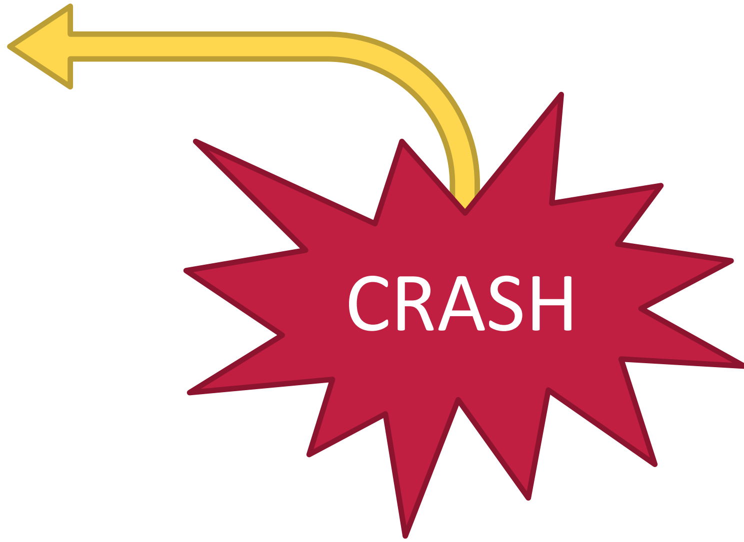
M memory module

C cache

RECOVERABLE MUTUAL EXCLUSION (RME) PROBLEM

loop forever

- Non-Critical Section (NCS)
- Recover
- Enter
- Critical Section (CS)
- Exit



Asynchronous, unreliable processes.

Golab and Ramaraju [PODC'16]

TERMINOLOGY

Passage:

Sequence of step taken by a process from when it begins Recover to when it completes Exit, or crashes, whichever occurs first.

Super-passage:

Maximal non-empty collection of consecutive passages executed by the same process where (only) the last passage in the collection is failure-free.

OBSERVATIONS

1. A process enters the CS at most once in each passage.
2. A process may enter the CS up to $f+1$ times in a super-passage where it fails f times.
3. A process must reenter the CS after it fails in Exit.

TWO WAYS TO NEST LOCKS

Way 1:

```
L1.lock()
L2.lock()
L2.unlock()
L1.unlock()
```

Way 2:

```
L1.lock()
L2.lock()
L1.unlock()
L2.unlock()
```

RME CORRECTNESS PROPERTIES

Mutual Exclusion (ME)

Deadlock Freedom (DF) ← revised

Starvation Freedom (SF) ← revised

Bounded Recovery (BR) ← new

Critical Section Re-entry (CSR) ← new

Golab and Ramaraju [PODC'16]

EXAMPLE OF REVISED PROPERTY

Starvation Freedom (SF):

For any infinite fair history *H*, if a process p_i leaves the non-critical section in some step of *H* then eventually p_i itself enters the CS, or else there are infinitely many crash steps in *H*.

EXAMPLE OF NEW PROPERTY

Critical Section Re-entry (CSR):

If a process *p* crashes inside the CS, then the next process to enter the CS is also *p*.

Property required for nesting locks correctly!

TEST-AND-SET LOCK

Shared variable: T, initially 0

Algorithm for process p_i:

Enter

loop while TestAndSet(T) = 1

back off

end loop

Critical Section

Exit

$T := 0$

Properties:

Mutual Exclusion

Deadlock Freedom

Wait-free Exit

Upper & lower bounds	single-word Read/Write/CAS…	+ single-word FAS/FAA/CAS
Mutual Exclusion	$O(\log N)$ Yang, Anderson [DC'95] $\Omega(\log N)$ Attiya, Hendler, Woelfel [STOC'08]	$O(1)$ Mellor-Crummey, Scott [TOCS'91]
Recoverable Mutual Exclusion	$O(\log N)$ Golab, Raramaju [PODC'16] Jayanti, Joshi [DISC'17] $\Omega(\log N)$ Attiya, Hendler, Woelfel [STOC'08]	$O(\log N / \log \log N)$ Golab, Hendler [PODC'17] Jayanti, Jayanti, Joshi [PODC'19] $O(1)$ Golab, Hendler [PODC'18]

AN IMPORTANT DIFFERENCE

O(log N / log log N)

Golab, Hendler [PODC'17]

Jayanti, Jayanti, Joshi [PODC’19]

← independent failures

O(1)

Golab, Hendler [PODC'18]

← system-wide failures

THOUGHTS ON STARVATION FREEDOM

Golab and Ramaraju [PODC'16] allow processes to starve in executions with infinitely many failures:

Starvation Freedom:

For any infinite fair history *H*, if a process p_i leaves the non-critical section in some step of *H* then eventually p_i itself enters the CS, or else there are infinitely many crash steps in *H*.

THOUGHTS ON STARVATION FREEDOM

Golab and Hendler [PODC'18] introduced an additional correctness property that mitigates this problem:

Failures-Robust Fairness (FRF):
For any fair history *H* containing infinitely many super-passages, if a process p_i leaves the NCS in some step of *H* then p_i eventually itself enters the CS.

THOUGHTS ON STARVATION FREEDOM

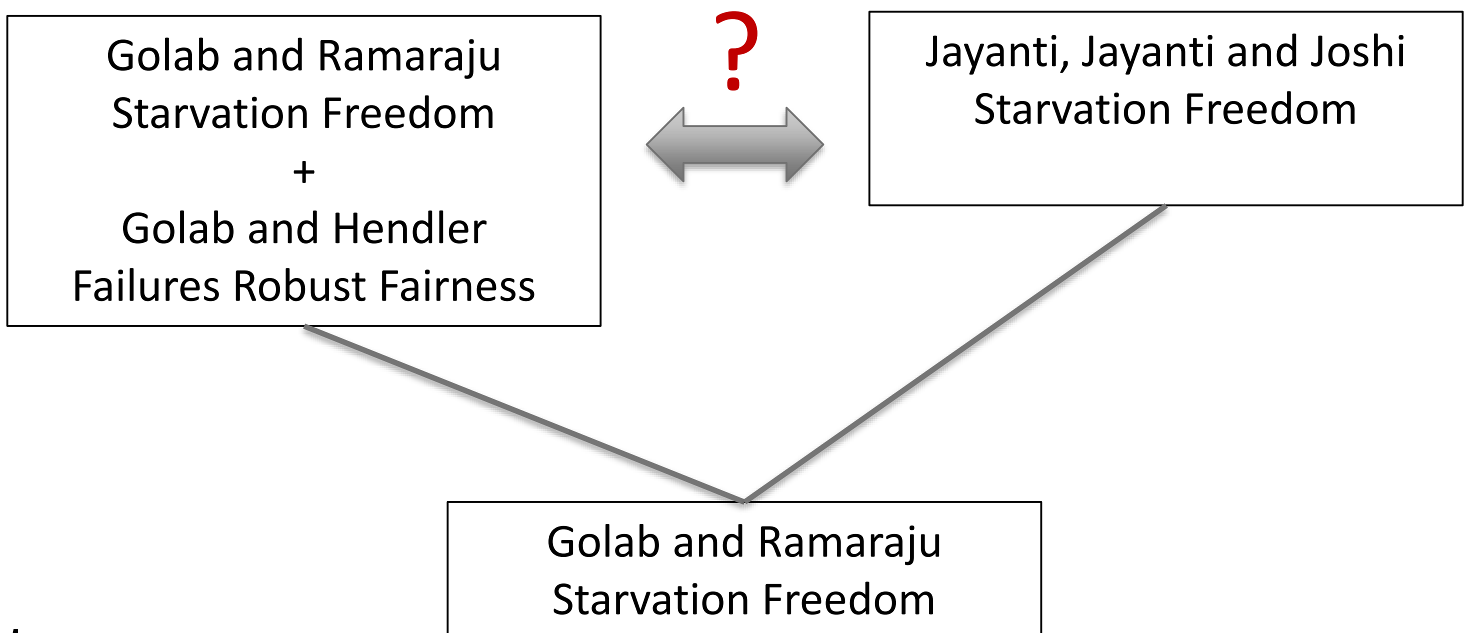
Jayanti, Jayanti, and Joshi [PODC'19] also proposed an alternative SF property:

Starvation Freedom:

If every process crashes only a finite number of times in each of its super-passages in a run, then every process enters the CS in each of its super-passages in that run.

RELATIONSHIP BETWEEN PROPERTIES

Strongest



Weakest

RECOVERABLE CONSENSUS

PROBLEM DEFINITION

Agreement: distinct processes never output different decisions.

Validity: each decision returned is the proposal value of some process.

Recoverable wait-freedom: each time a process executes it algorithm from the beginning, it either returns a decision after a finite number of its own steps, or crashes.

CONSENSUS HIERARCHY

Type	Consensus Number
Compare-And-Swap	∞
Test-And-Set Fetch-And-Store Fetch-And-Add Stack Queue	2
Read/Write Register	1

Herlihy, 1991

RECOVERABLE CONSENSUS HIERARCHY:
SYSTEM-WIDE FAILURES

Type	R-Consensus Number
Compare-And-Swap	∞
Test-And-Set Fetch-And-Store Fetch-And-Add Queue Stack	2
Read/Write Register	1

TRANSFORMATION

Shared variables:

- $P[1..2]$: **array** of proposal values, **init** $\perp$
- C: conventional wait-free 2-process consensus object
- D: decision, **init** $\perp$

Private variables:

- other: process ID
- d: decided value

OBSERVATIONS

If a process begins executing *C* and then crashes, it cannot execute *C* again!

If a process knows that it lost, then it also knows exactly who won.

TRANSFORMATION

Procedure Decide(v: proposal value) for proc. p_i, $i \in 1..2$

if $i = 1$ **then** other := 2 **else** other := 1
if $P[i] = \bot \wedge P[\text{other}] = \bot$ **then**
    $P[i] := v$
    $d := C.$**Decide**(v)
    $D := d$
    return d
else if $D \neq \bot$ **then**
    return D
else if $P[i] \neq \bot \wedge P[\text{other}] = \bot$ **then**
    return $P[i]$
else if $P[i] = \bot \wedge P[\text{other}] \neq \bot$ **then**
    return $P[\text{other}]$
else // $P[i] \neq \bot \wedge P[\text{other}] \neq \bot$
    return $P[1]$

RECOVERABLE CONSENSUS HIERARCHY: ≤F INDEPENDENT FAILURES

Type	R-Consensus Number
Compare-And-Swap	∞
Test-And-Set Fetch-And-Store Fetch-And-Add Queue Stack	2
Read/Write Register	1

TRANSFORMATION

Shared variables:

- $R[1..n]$: **array** of read/write register, **init** 0
- $C[0..f]$: **array** of conventional wait-free n-process consensus objects
- $D[0..f]$: **array** of read/write register, **init** $\bot$

Private variables:

- k, k': integers, uninitialized
- d: decision value, uninitialized

f = upper bound on total number of failures

TRANSFORMATION

Procedure Decide(v: proposal value) for proc. p_i, $i \in 1..n$

```
for k in 0..f do
    if R[i] = k then
        R[i] := k + 1
        // check for a decision in a
           lower-numbered iteration
        for k′ ∈ 0..(k − 1) do
            if D[k′] ≠ ⊥ then
                v := D[k′]
        d := C[k].Decide(v)
        D[k] := d
        // check for a collision with a
           higher-numbered iteration
        if k < f then
            for z ∈ 1..n, z ≠ i do
                if R[z] > R[i] then
                    d := ⊥
        // return decision if known
        if d ≠ ⊥ then
            return d
```

RECOVERABLE CONSENSUS HIERARCHY: INDEPENDENT FAILURES

Type	R-Consensus Number
Compare-And-Swap	∞
Test-And-Set	1
Read/Write Register	1

IMPOSSIBILITY RESULTS

Result 1: space bound

If there are up to f failures then f +1 instances of TAS are necessary.

Result 2: consensus number

If there are arbitrarily many failures, then recoverable consensus is not solvable even with infinitely many TAS objects!

RESULT 1: SPACE BOUND

Valency argument, similar to Herlihy’s but modified for crash-recovery failures.

v*-potent state *s: there exists a sequence of steps starting from *s* such that some process returns *v*.

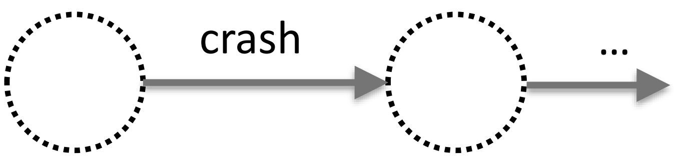
***v*-valent state *s*:** *v*-potent but not *v′*-potent for any $v' \neq v$.

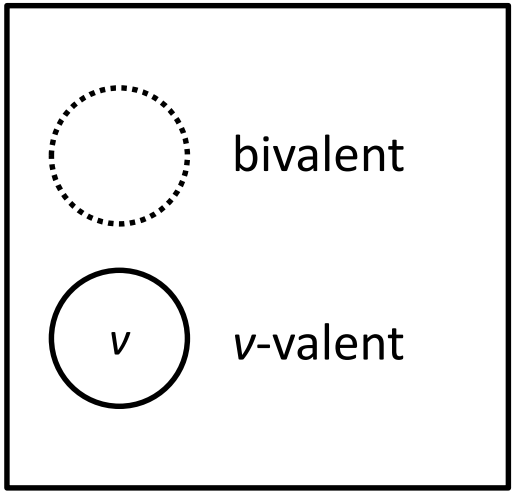
univalent state *s*: *v*-valent for some *v*.

bivalent state: both *v*-potent and *v′*-potent for some distinct values *v* and *v′.*

RESULT 1: SPACE BOUND

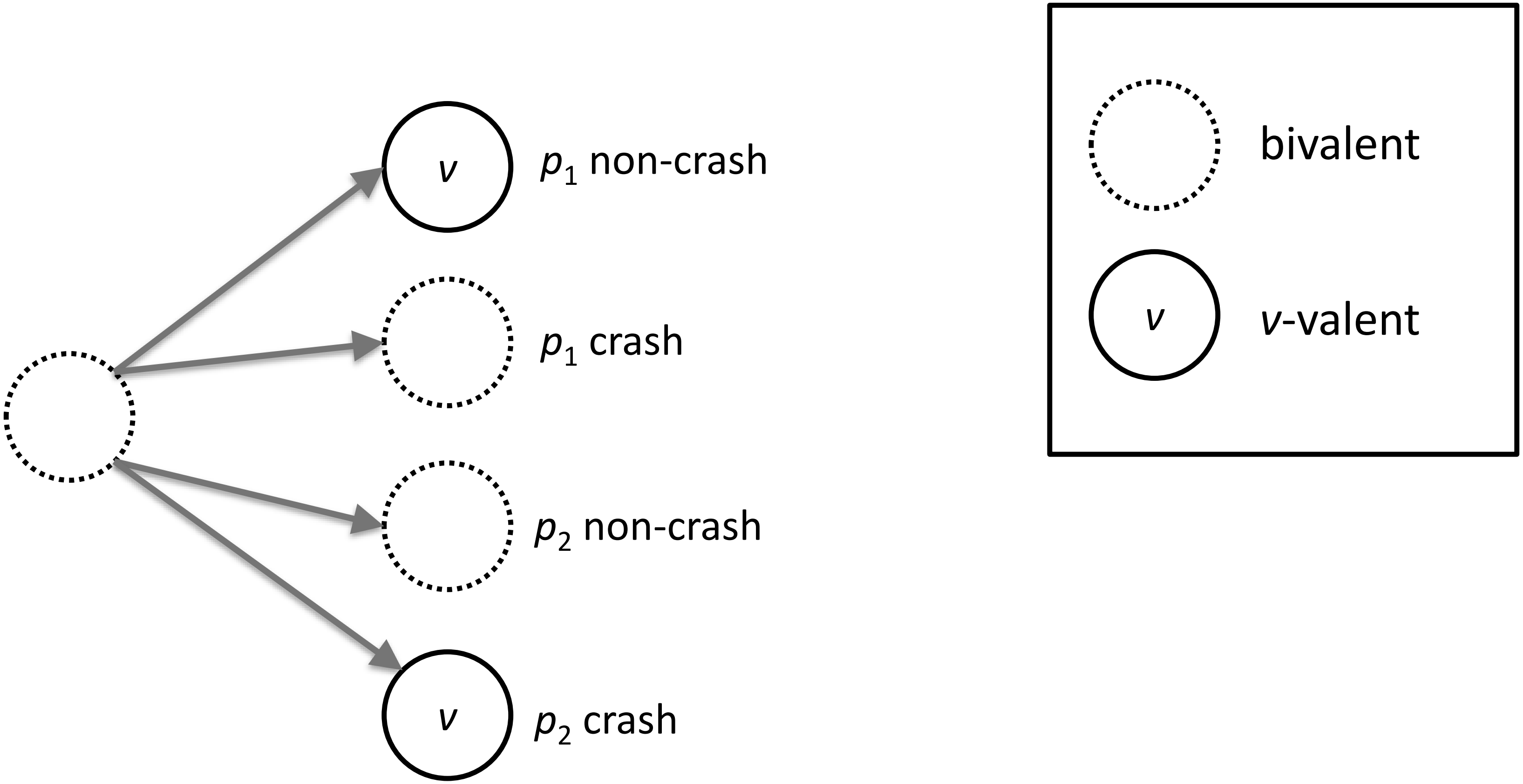
Why Herlihy's technique breaks:





RESULT 1: SPACE BOUND

Why Herlihy's technique breaks:



RESULT 1: SPACE BOUND

Consider a subset of execution histories satisfying the following invariants:

1. Only one designated process p_i is permitted to fail.
2. If p_i fails f times then it has touched f distinct TAS objects.

RESULT 1: SPACE BOUND

When is p_i allowed to fail?

Only if its previous step was its first access to some TAS object in the execution.

RESULT 2: CONSENSUS NUMBER

Consider a subset of execution histories satisfying the following invariants:

1. Only one designated process p_i is permitted to fail.
2. If p_i fails f times then the other process p_j has taken at least f steps failure-free.

Note: A similar proof technique was developed in parallel by Attiya, Ben-Baruch, and Hendler for NRL [PODC'18].

RESULT 2: CONSENSUS NUMBER

When is p_i allowed to fail?

Only if its previous step was its first access to some TAS object in the execution, and moreover that object was also accessed by the other process p_j.

TAKE-AWAYS

HOW WE GOT HERE

Talked about persistent memory, thought about crash-recover failures.

RESEARCH DIRECTIONS

1. Prove a tight RMR complexity bound for RME with independent failures and single-word primitives.
2. Devise alternative O(1)-RMR solutions for RME with simultaneous failures.
3. Establish a more precise relationship between the conventional consensus hierarchy and the recoverable consensus hierarchy.