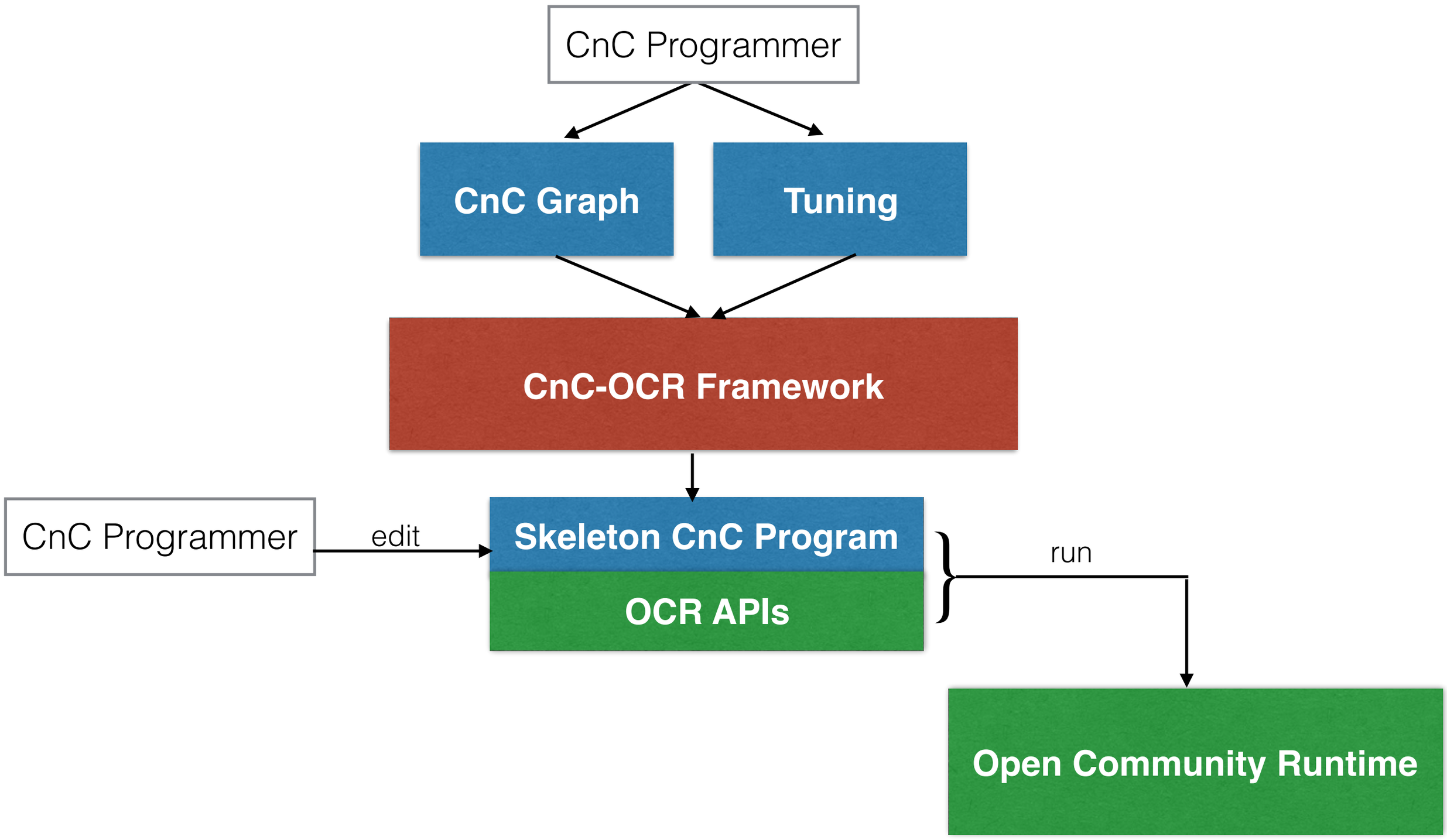
Towards Resilient CnC-OCR

Sara S. Hamouda*, Sanjay Chatterjee**, Nick Vrvilo**,
Zoran Budimlic**, Vivek Sarkar**

*The Australian National University,
**Rice University

CnC-2016: The 8th Annual Concurrent Collections Workshop
Rochester, New York, 27 September 2016

CnC-OCR Framework



Reliability of Large Scale Systems

- As HPC systems grow larger, their failure rates increase[1].

- Exa-scale systems are expected to have mean time between failures (MTBF) in terms of minutes[2].

[1] B. Schroeder and G. A. Gibson, “Understanding failures in petascale computers,” in *Journal of Physics: Conference Series*, vol. 78, no. 1. IOP Publishing, 2007
[2] G. Zheng, L. Shi, and L. V. Kalé, “FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI,” in IEEE International Conference on Cluster Computing, pp. 93–103, IEEE, 2004.

Problem

- Resilience support is missing from all CnC and OCR implementations.

Problem

- Resilience support is missing from all CnC and OCR implementations.

- **Objective:** allow large scale CnC-OCR applications to efficiently recover from process failures.

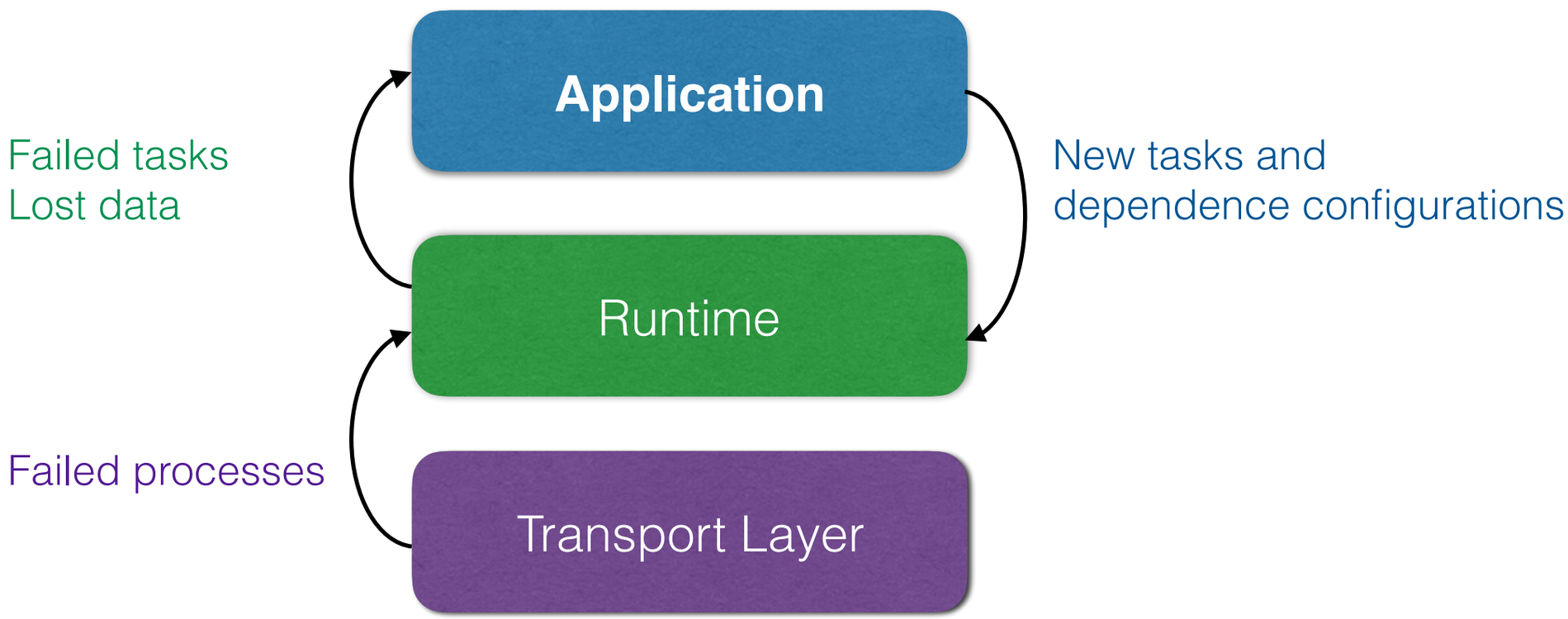
Problem

- Resilience support is missing from all CnC and OCR implementations.
- **Objective:** allow large scale CnC-OCR applications to efficiently recover from process failures.
- **Solution:**
  - Extend OCR with user-level fault tolerance.
  - Implement failure recovery algorithm for CnC-OCR applications using OCR fault tolerance support

Agenda

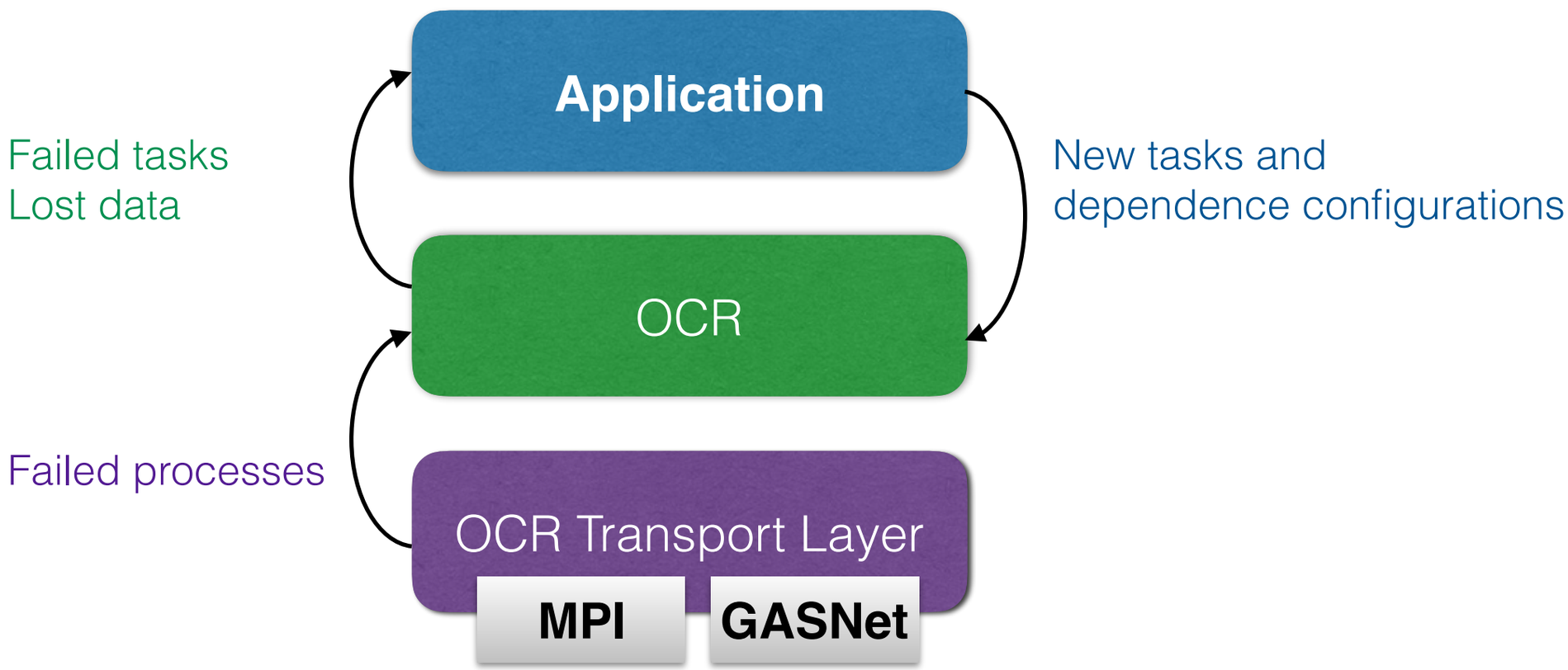
- OCR User-Level Fault Tolerance
  - Failure Detection
  - Failure Propagation
- CnC-OCR Application Recovery

User-Level Fault Tolerance



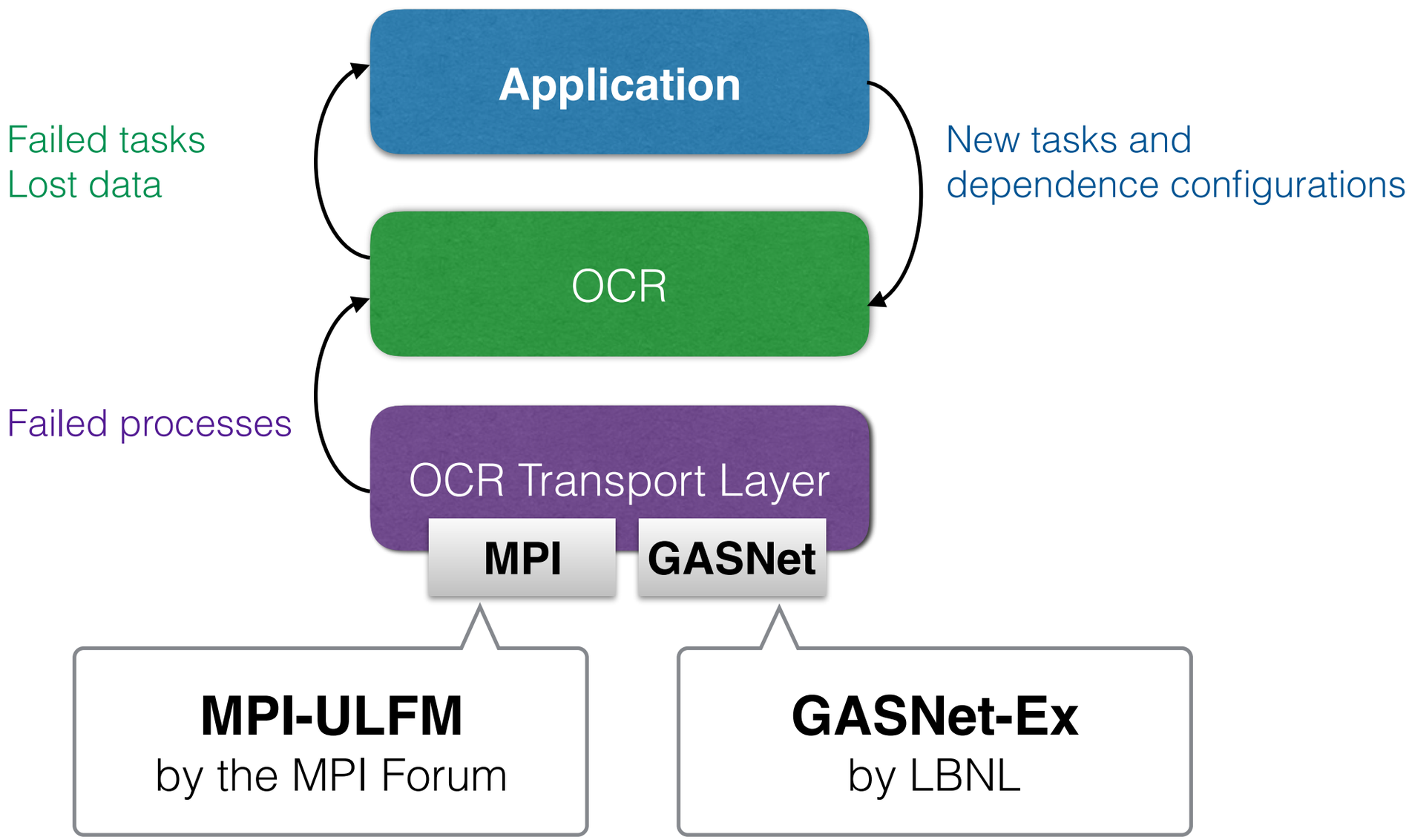
User-Level Fault Tolerance

1) Failure Detection



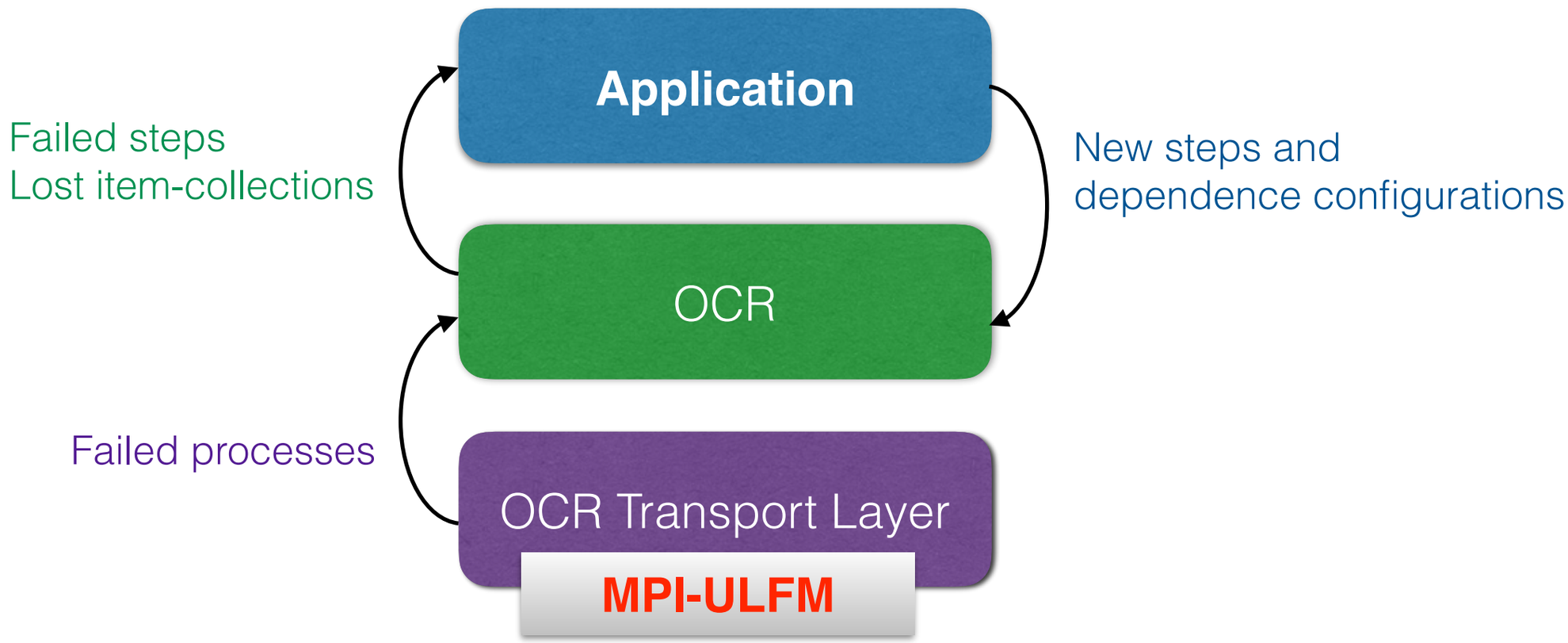
User-Level Fault Tolerance

1) Failure Detection



User-Level Fault Tolerance

1) Failure Detection



MPI User-Level Failure Mitigation

MPI User-Level Failure Mitigation

MPI-3

Specified runtime behavior
after process failure

New MPI Operations

MPI User-Level Failure Mitigation

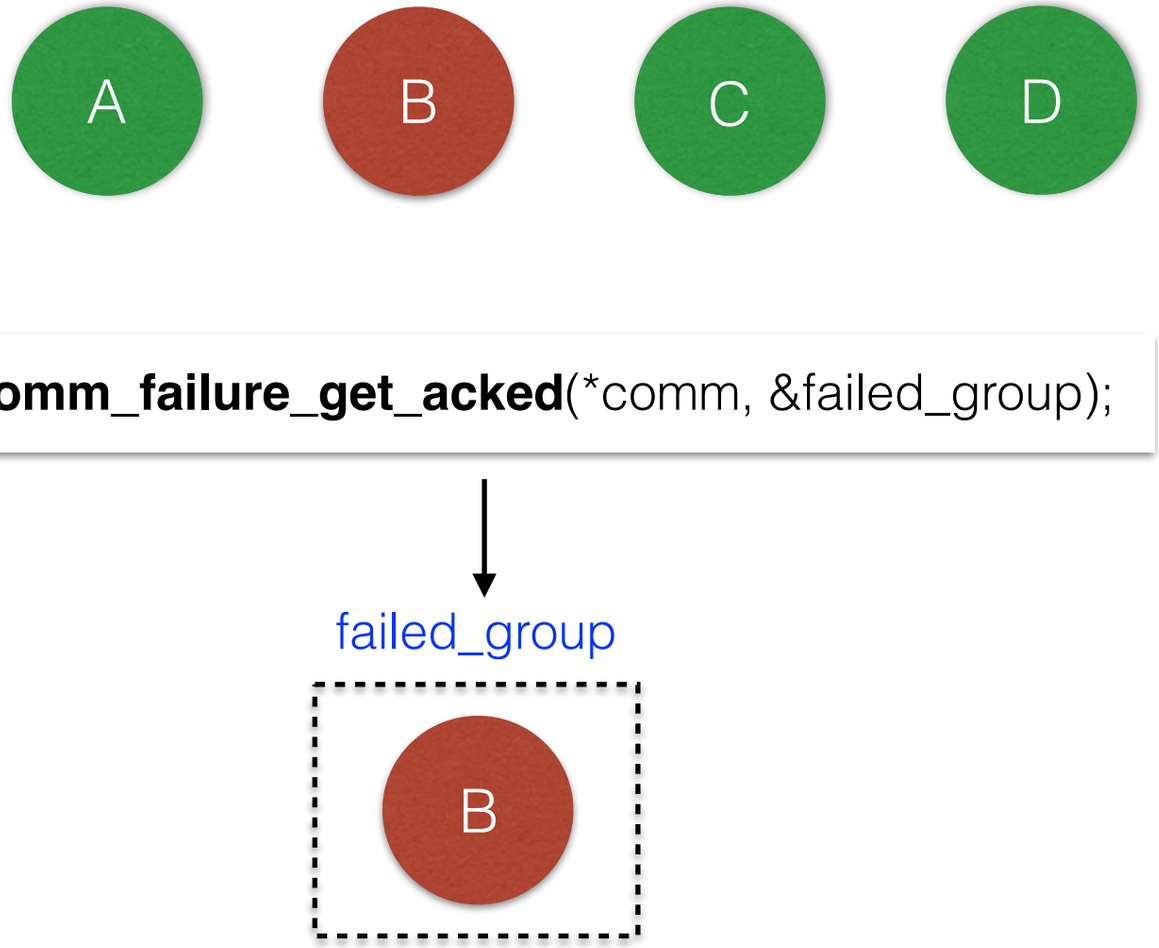
- Assumes **fail-stop** process failure.
- Allows **non-failed processes** to communicate.
- Failure reporting via **special error codes**.

MPI User-Level Failure Mitigation

- New MPI Operations:
  - MPI_Comm_failure_get_acked
  - MPI_Comm_revoke
  - MPI_Comm_shrink
  - MPI_Comm_agree

MPI User-Level Failure Mitigation

- New MPI Operations:
  - MPI_Comm_failure_get_acked
  - MPI_Comm_revoke
  - MPI_Comm_shrink
  - MPI_Comm_agree

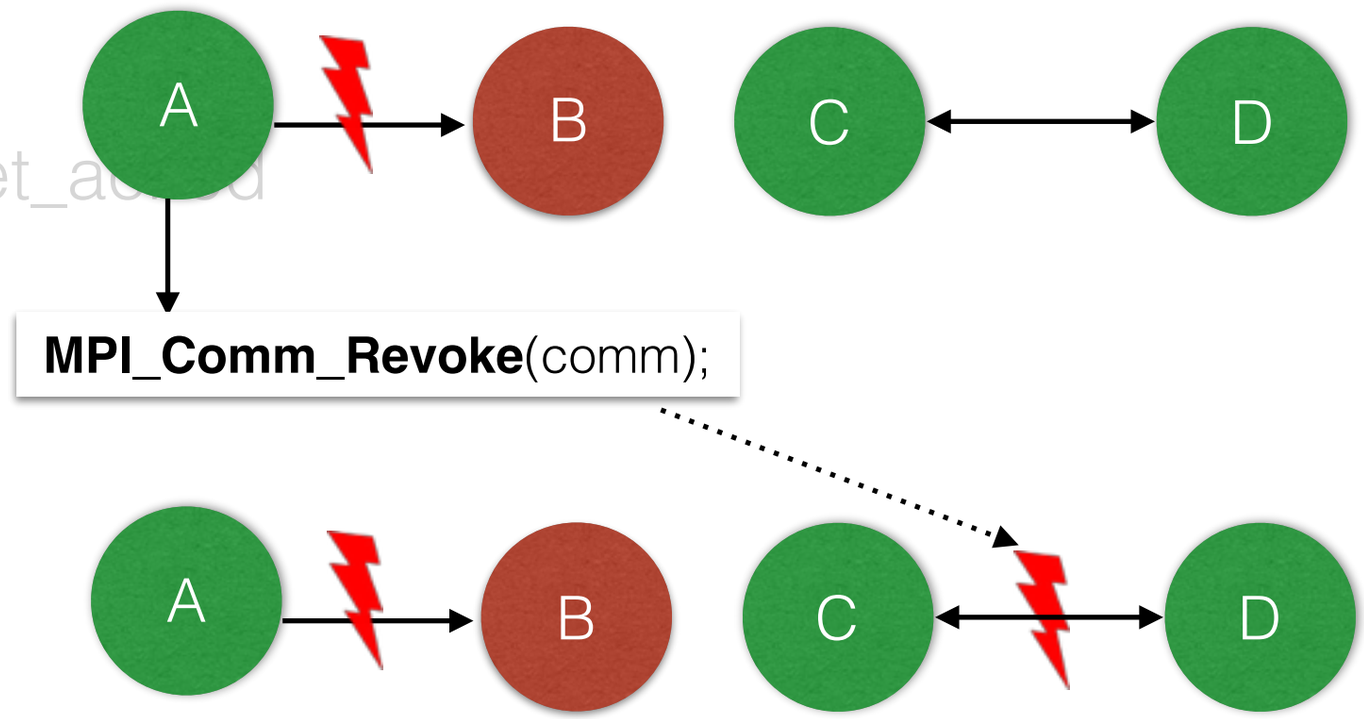


MPI User-Level Failure Mitigation

- New MPI Operations:
  - MPI_Comm_failure_get_acked
  - MPI_Comm_revoke
  - MPI_Comm_shrink
  - MPI_Comm_agree

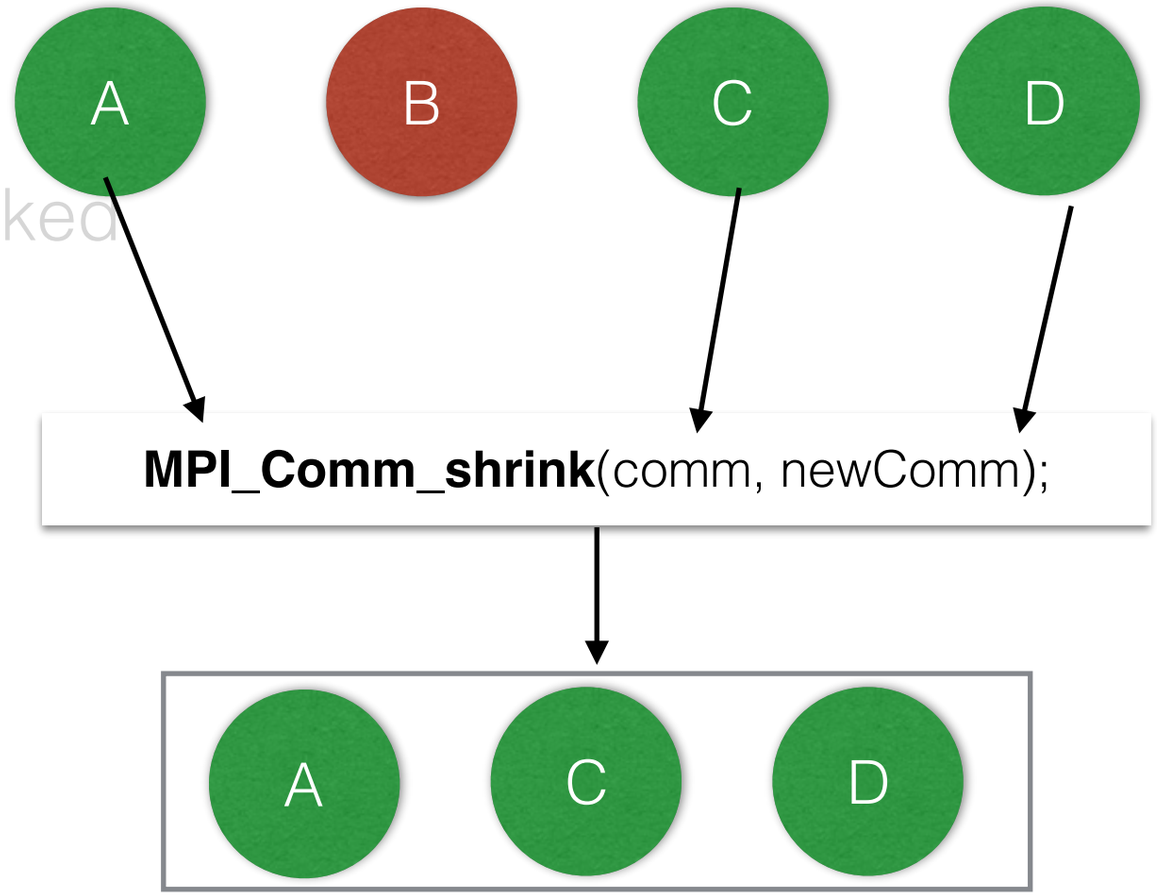
MPI User-Level Failure Mitigation

- New MPI Operations:
  - MPI_Comm_failure_get_acked
  - MPI_Comm_revoke
  - MPI_Comm_shrink
  - MPI_Comm_agree



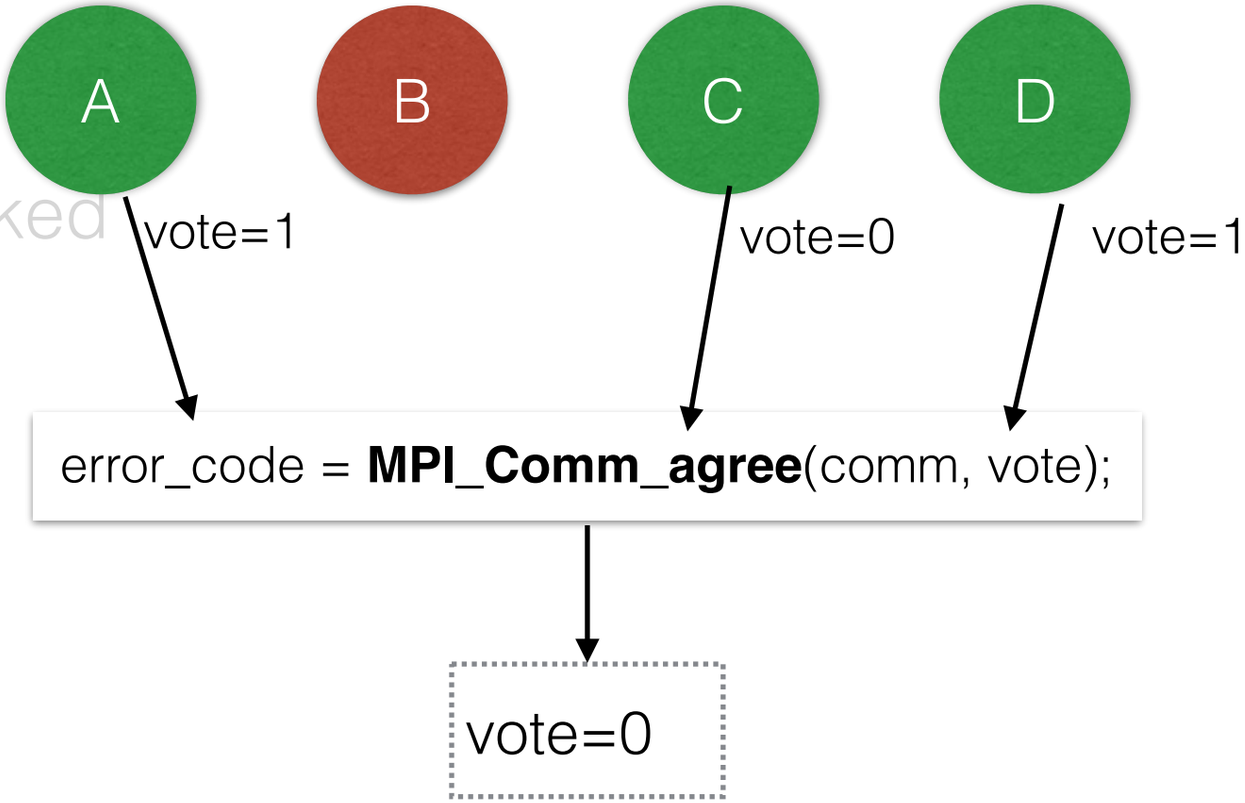
MPI User Level Failure Mitigation

- New MPI Operations:
  - MPI_Comm_failure_get_acked
  - MPI_Comm_revoke
  - MPI_Comm_shrink
  - MPI_Comm_agree



MPI User Level Failure Mitigation

- New MPI Operations:
  - MPI_Comm_failure_get_acked
  - MPI_Comm_revoke
  - MPI_Comm_shrink
  - MPI_Comm_agree



OCR Failure Detection with ULFM

- Failure detection:
  - `MPI_Isend / MPI_Irecv / MPI_Test`
  - `MPI_Iprobe (MPI_ANY_SOURCE, …)`

OCR Failure Detection with ULFM

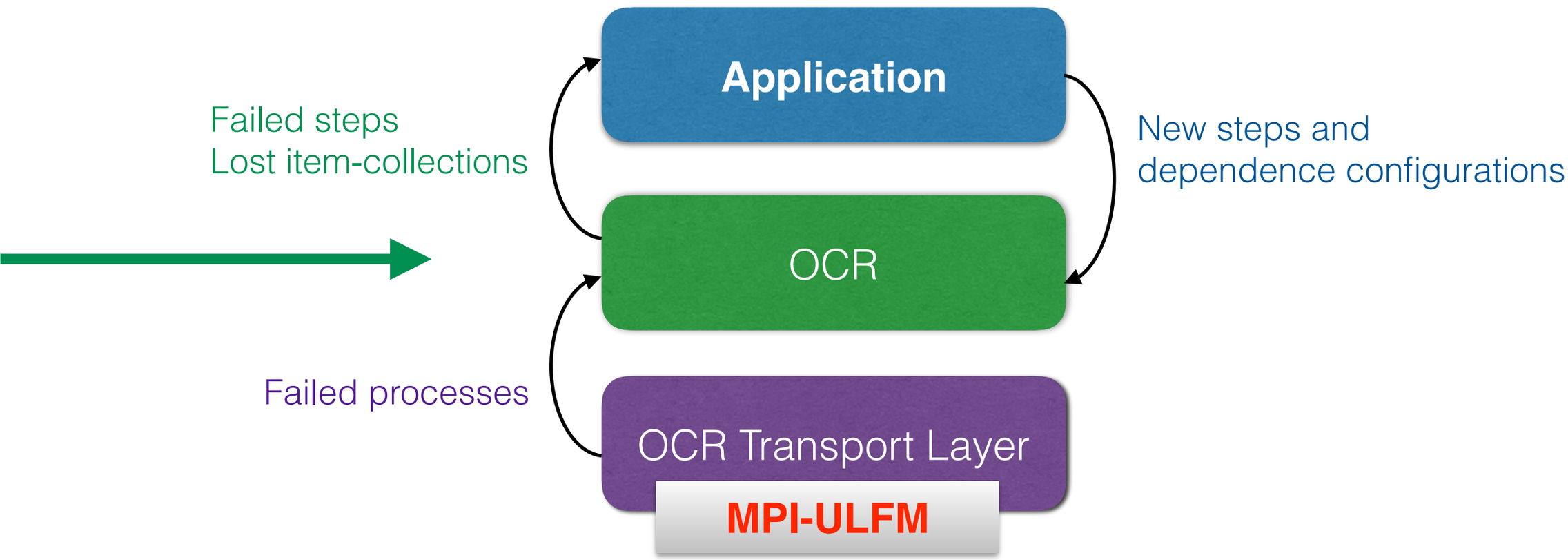
- Failure detection:
  - `MPI_Isend / MPI_Irecv / MPI_Test`
  - `MPI_Iprobe (MPI_ANY_SOURCE, …)`

OCR Failure Detection with ULFM

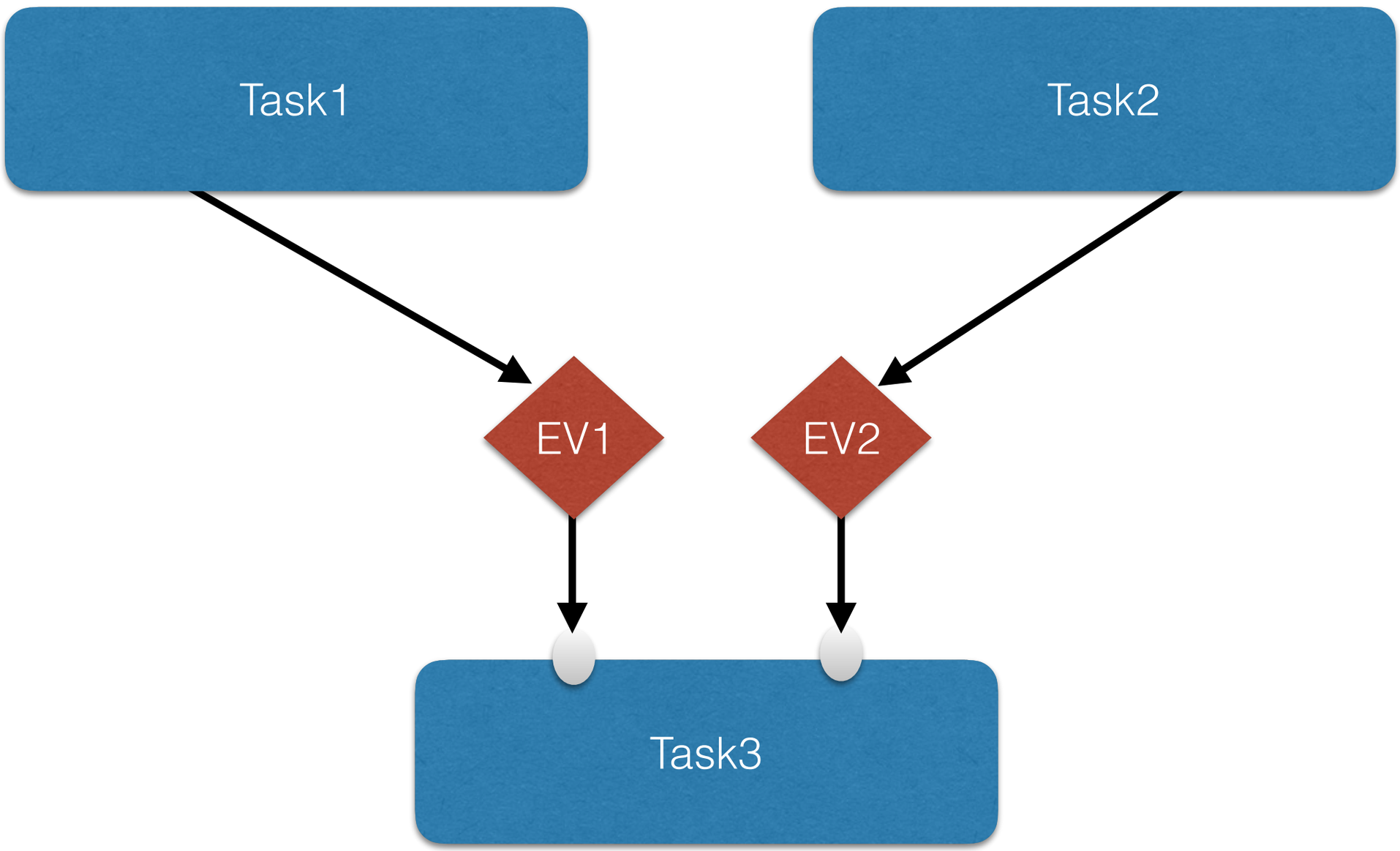
- Failure detection:
  - `MPI_Isend / MPI_Irecv / MPI_Test`
  - `MPI_Iprobe (MPI_ANY_SOURCE, …)`
- MPI_COMM_WORLD error handler:
  - Discovers dead processes (`MPI_Comm_failure_get_acked`)
  - Does **not** shrink or revoke the communicator.
  - Calls OCR->updateDeadLocations(…)

User-Level Fault Tolerance

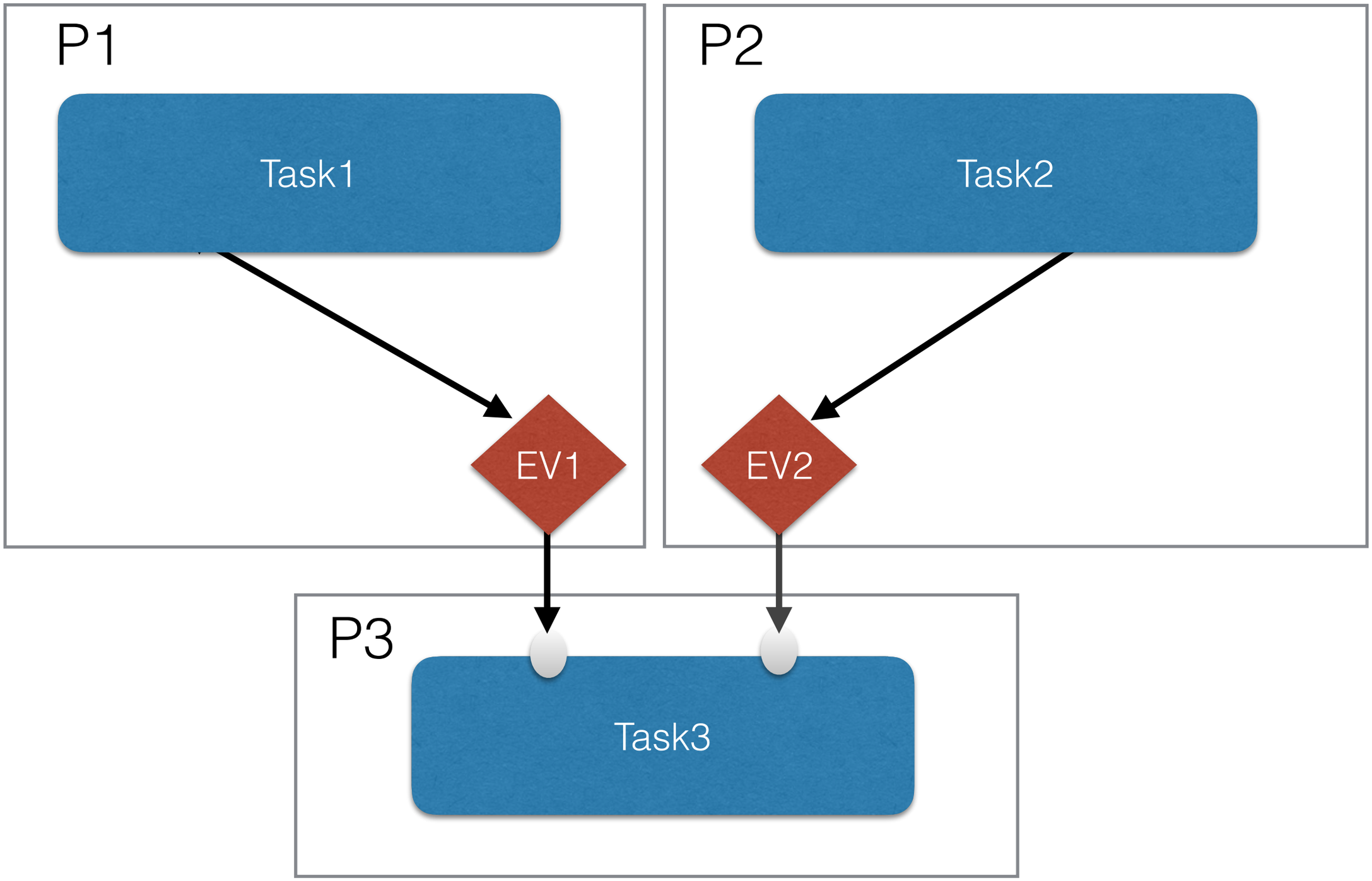
2) Failure Propagation



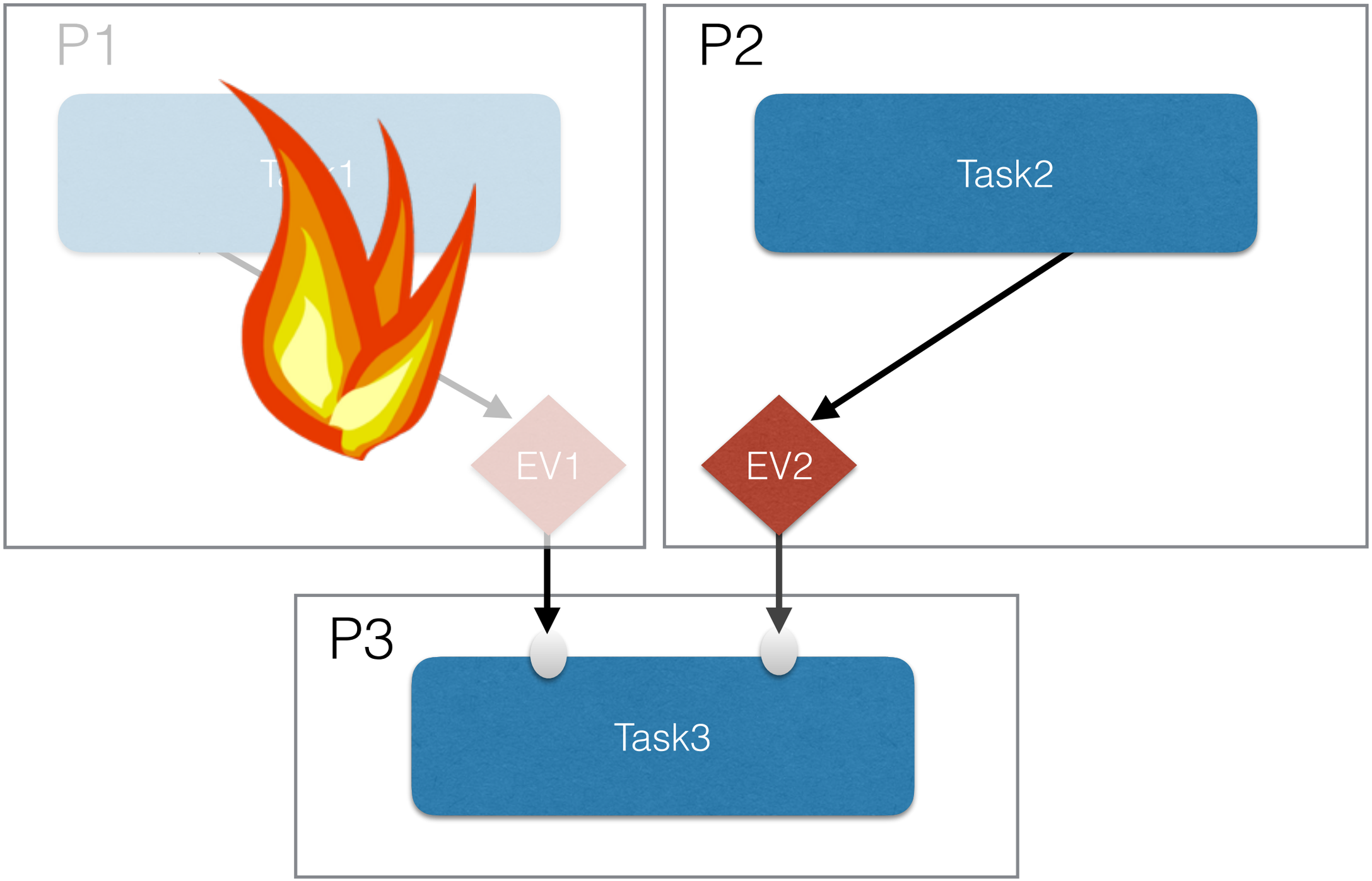
Example OCR Dependence Graph



Remote Dependence



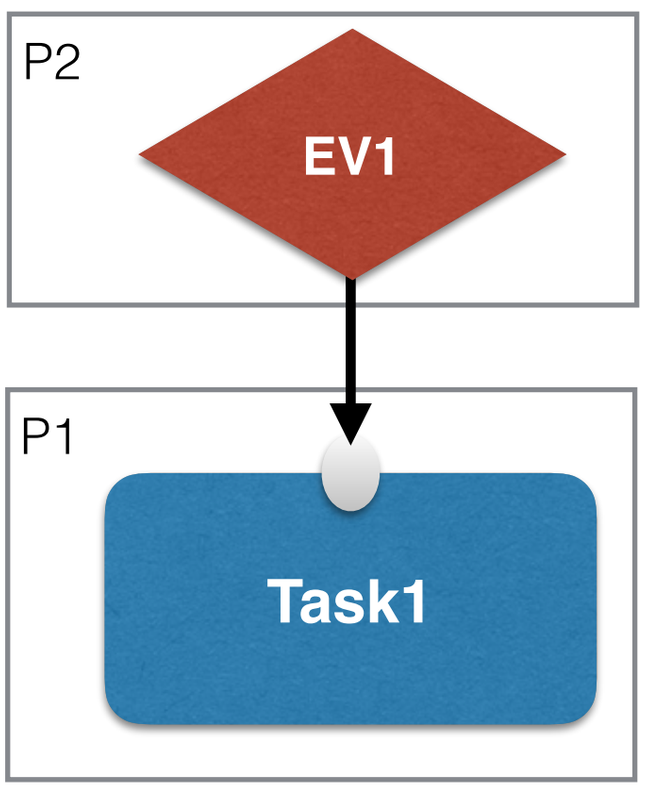
Remote Dependence



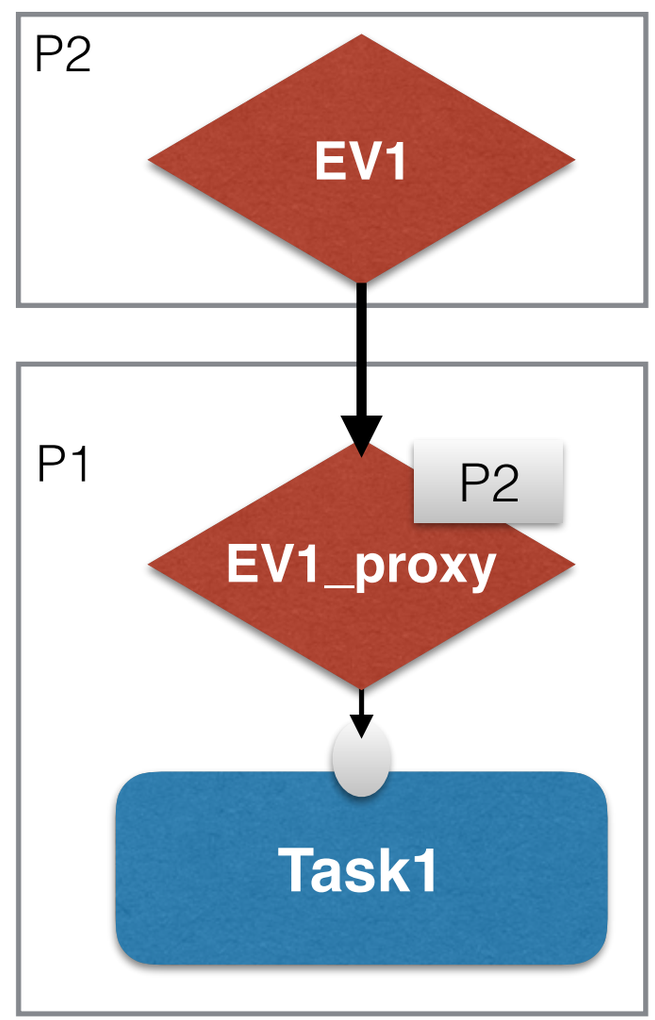
Detecting Lost Dependence

- When adding a dependence on a remote event, add a **local proxy event**

Non Resilient

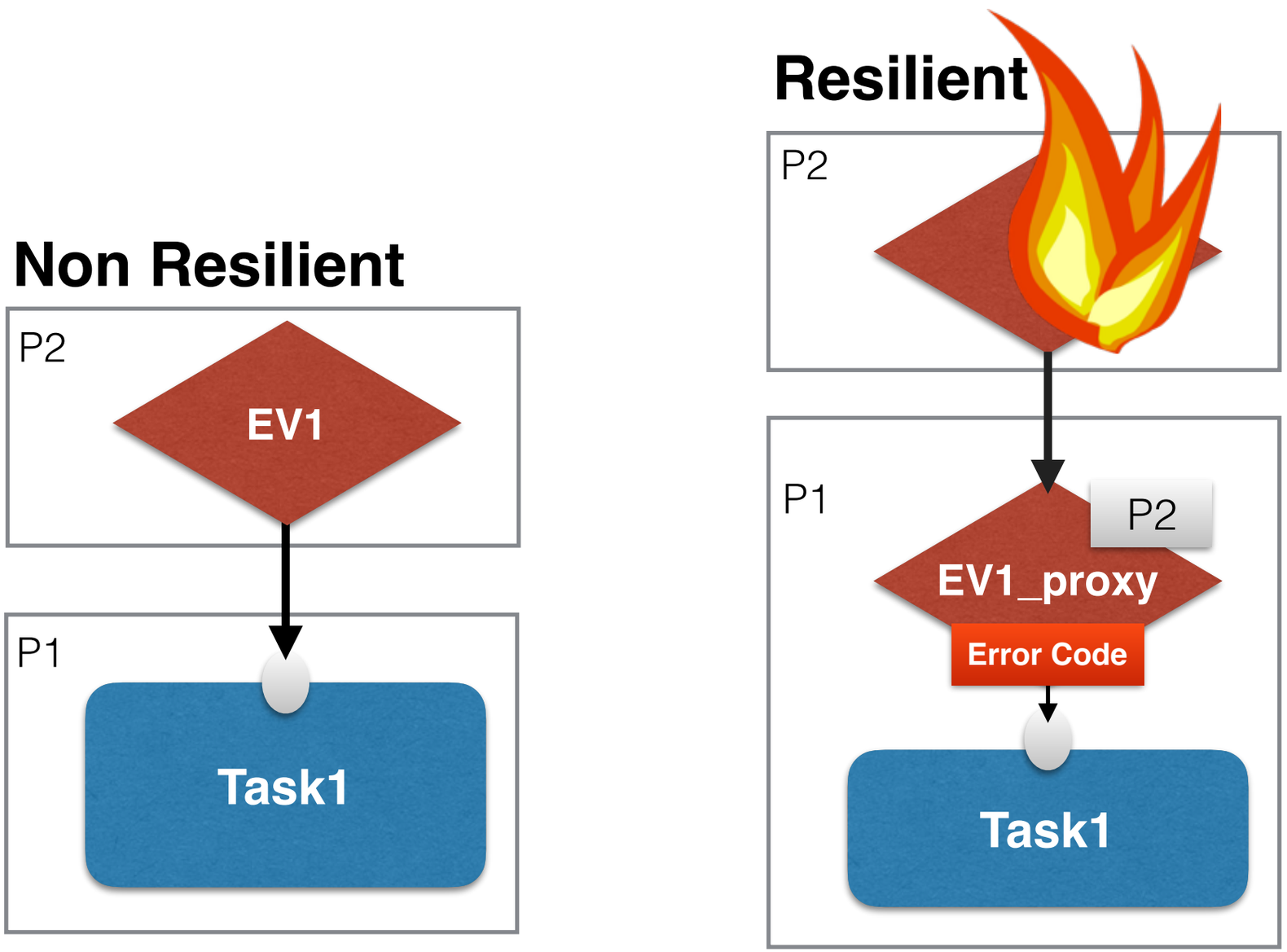


Resilient



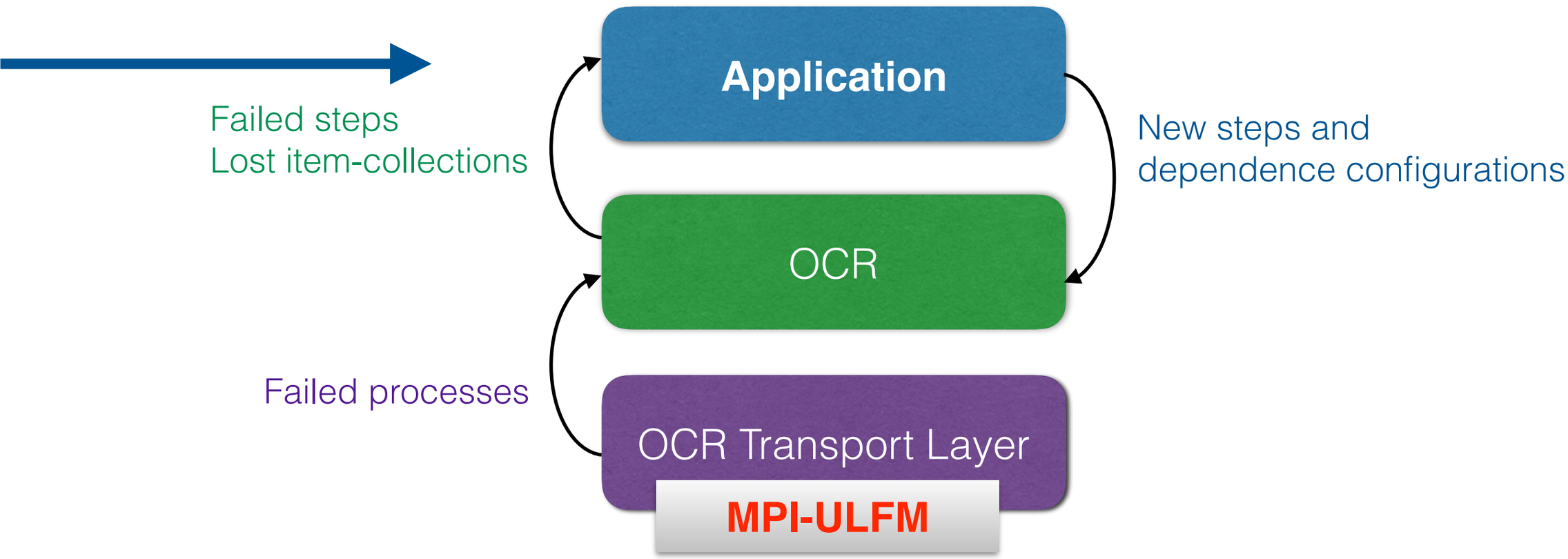
Detecting Lost Dependence

- When adding a dependence on a remote event, add a **local proxy event**

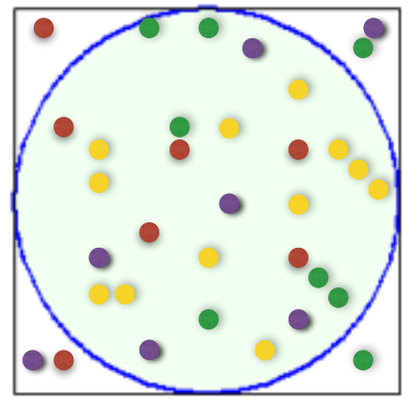


User-Level Fault Tolerance

3) Application Recovery

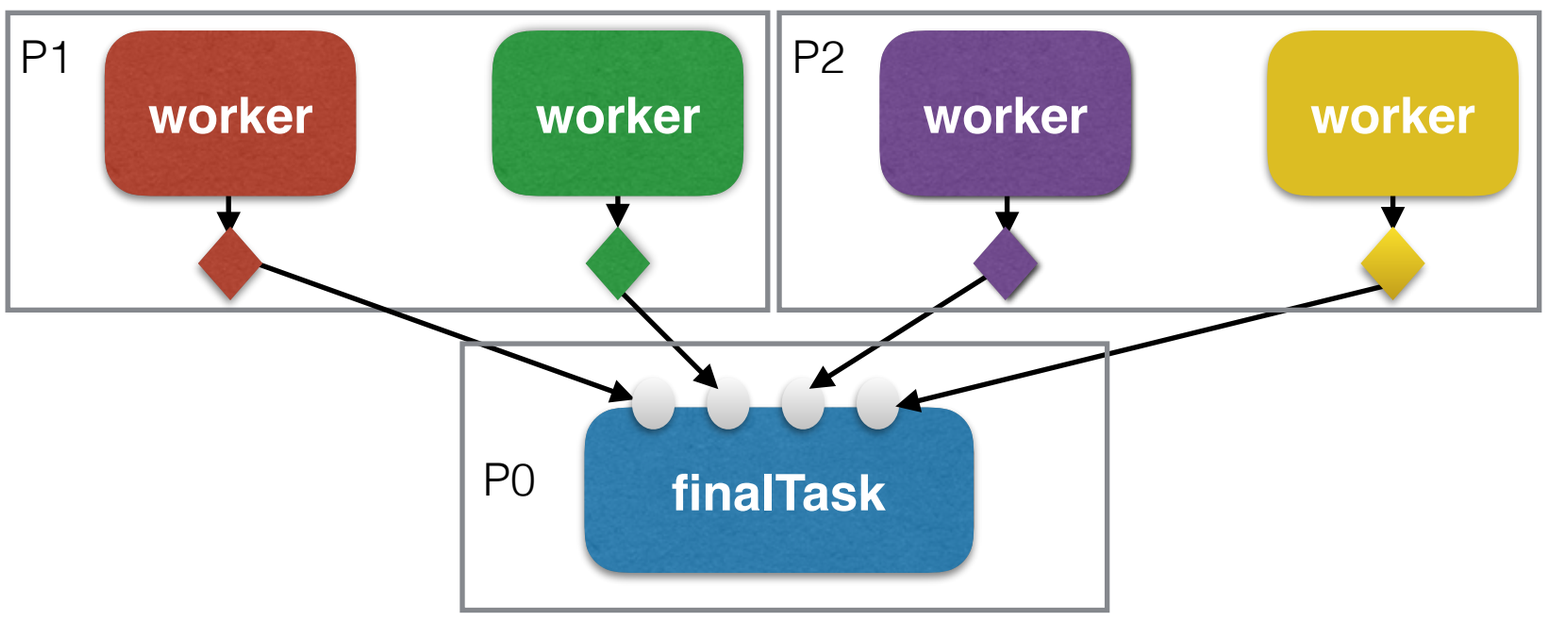


Monte Carlo PI



worker:

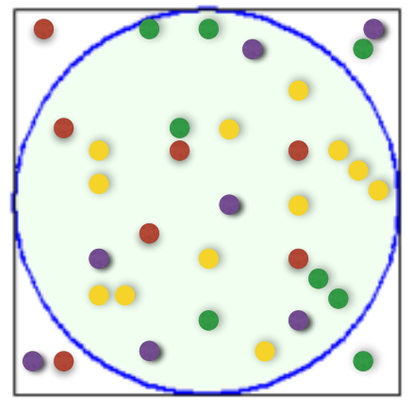
1. generate random points
2. count all generated points
3. count points inside circle
4. send counters to accumulator



finalTask:

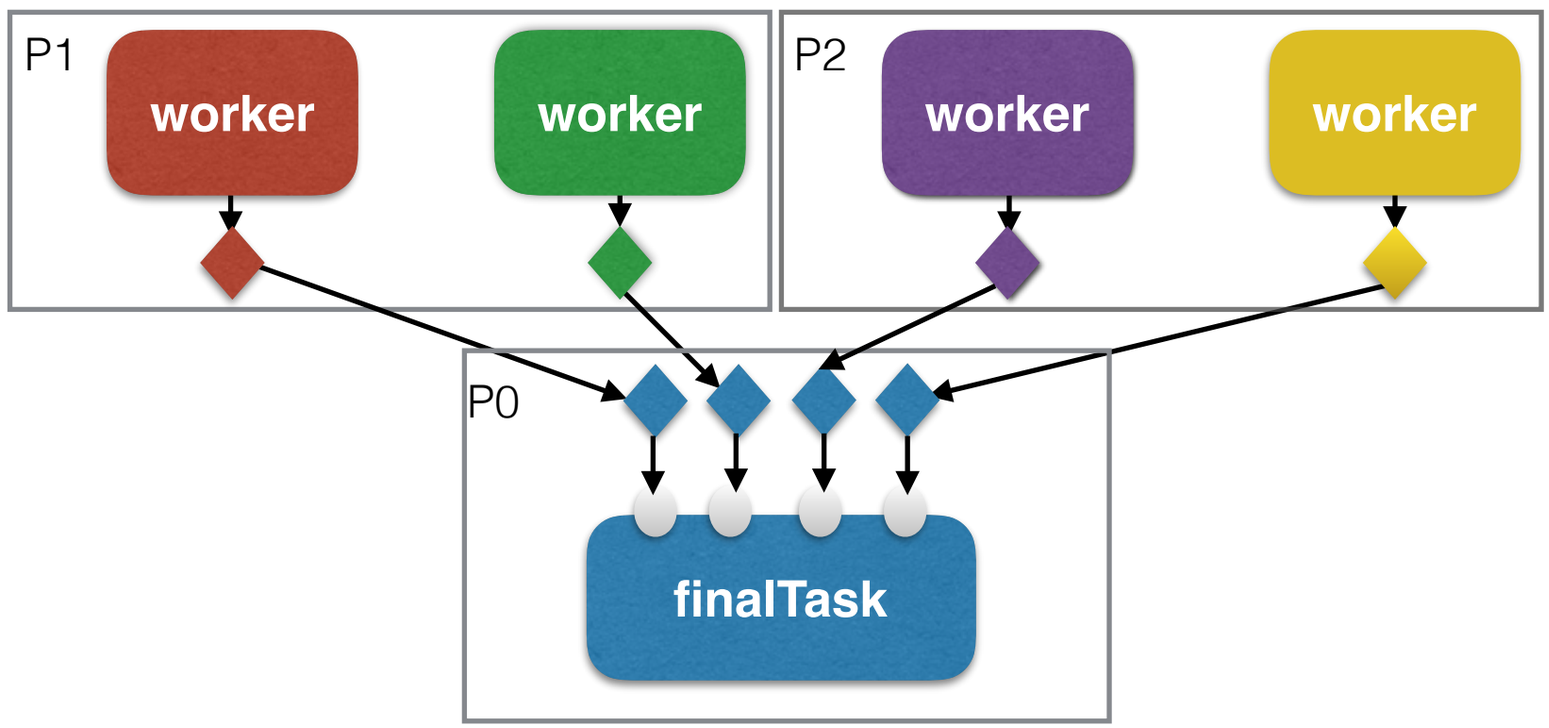
```
PI = 4 * all_points_inside_circle / all_points
```

Monte Carlo PI



worker:

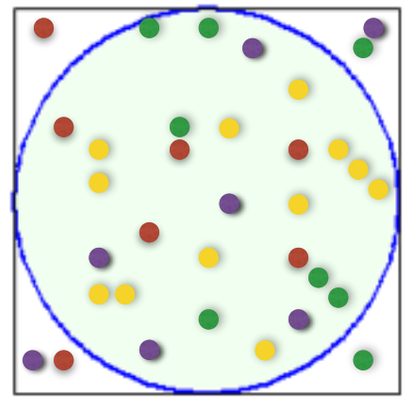
1. generate random points
2. count all generated points
3. count points inside circle
4. send counters to accumulator



finalTask:

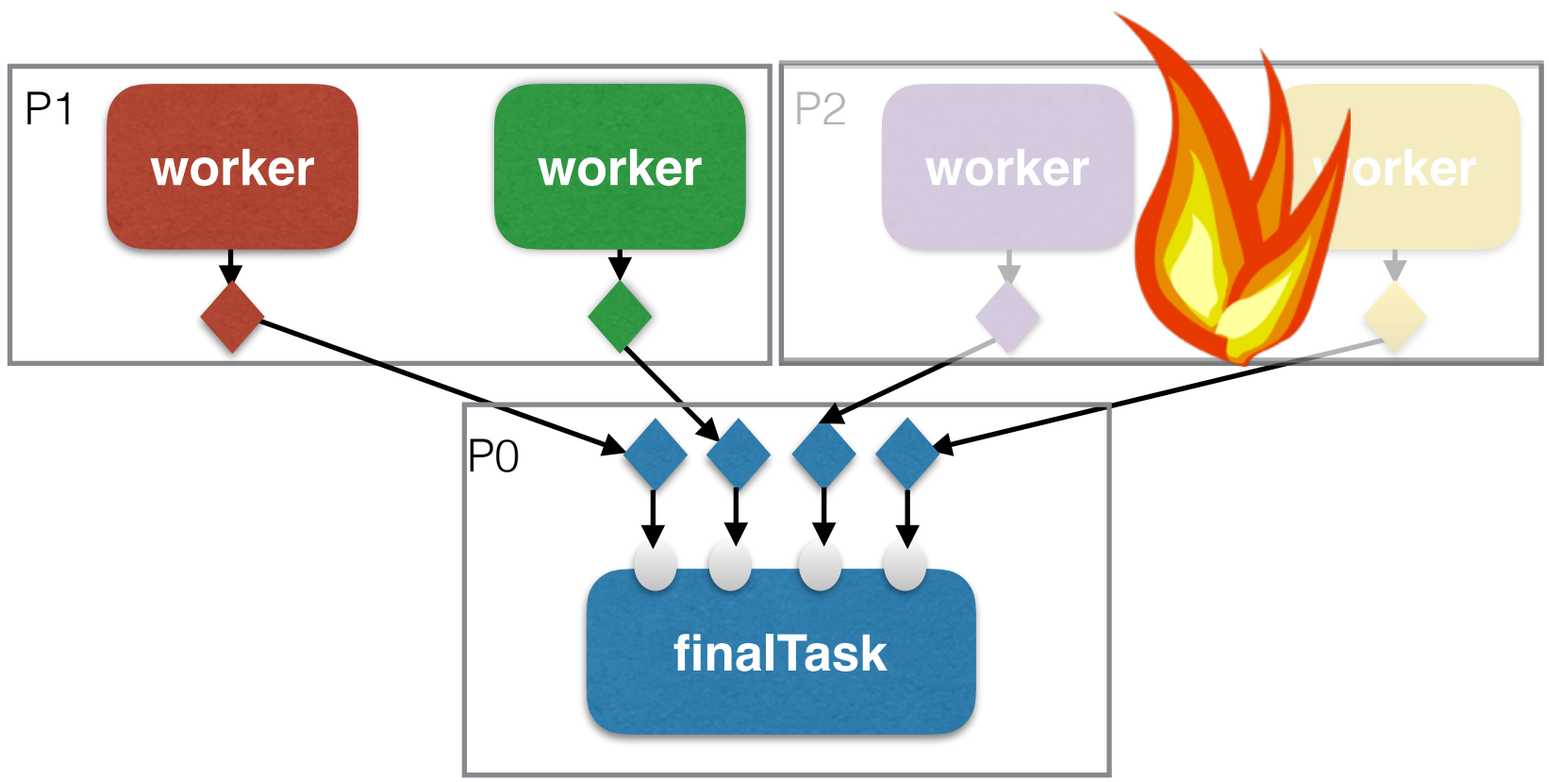
```
PI = 4 * all_points_inside_circle / all_points
```

Monte Carlo PI



worker:

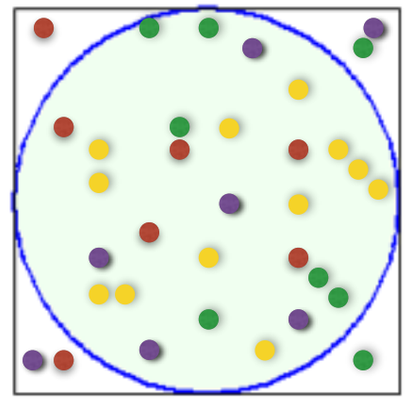
1. generate random points
2. count all generated points
3. count points inside circle
4. send counters to accumulator



finalTask:

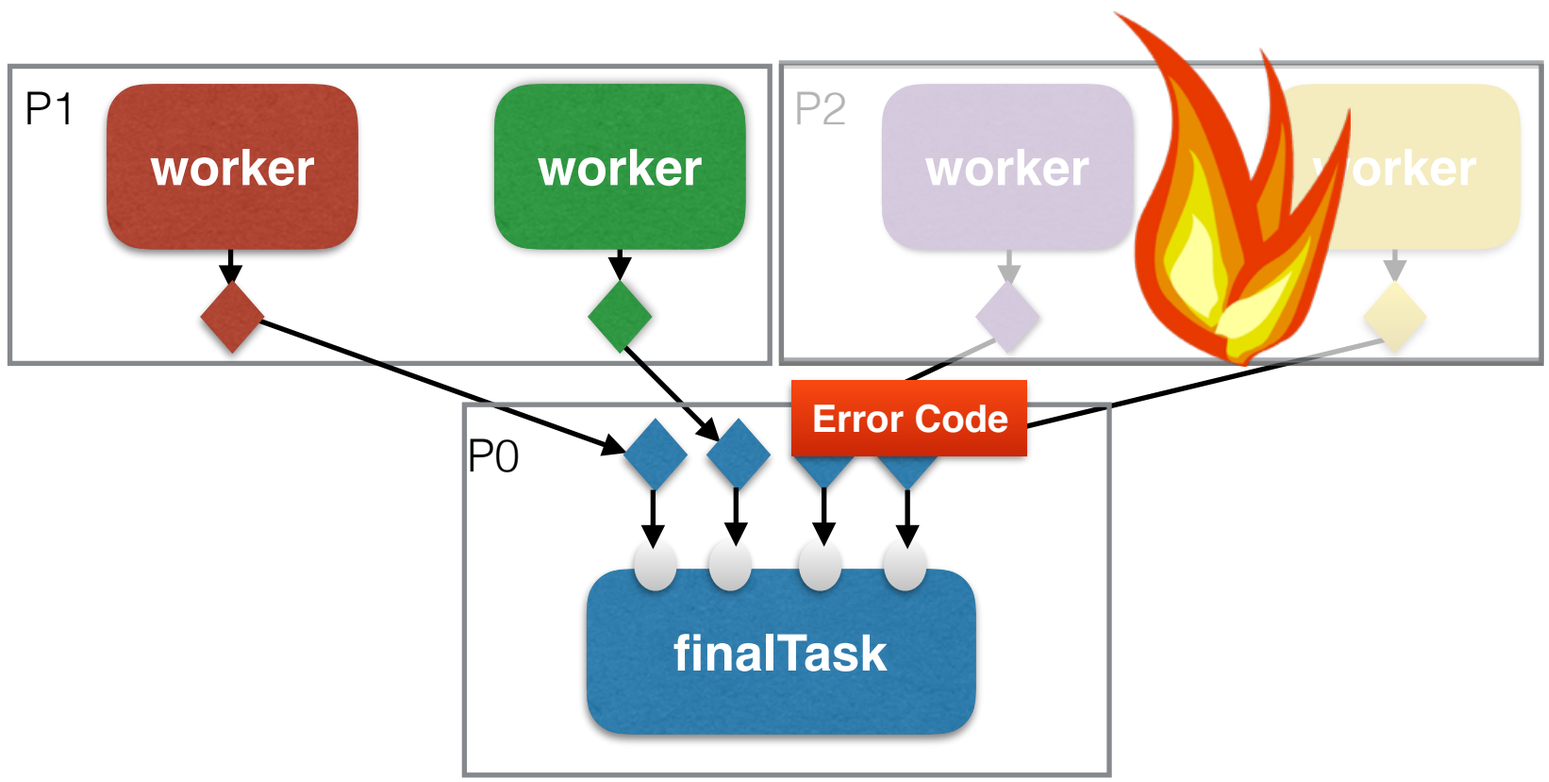
```
PI = 4 * all_points_inside_circle / all_points
```

Monte Carlo PI



worker:

1. generate random points
2. count all generated points
3. count points inside circle
4. send counters to accumulator



finalTask:

```
PI = 4 * all_points_inside_circle / all_points
```

Monte Carlo PI

```
1  void finalTask(cncTag_t i, WorkerData *data1, WorkerData*data2,
                              WorkerData *data3, WorkerData *data4)
2  {
3     WorkerData* dep[4] = {data1, data2, data3, data4};
4     u32 nPoints = 0;
5     u32 nWorkers = 0;
6     float PI = 0.0;
7     for (int i = 0 ; i < 4; i++) {
8         if (dep[i].valid) {
9             nPoints += dep[i].points;
10            nWorkers ++;
11        }
12    }
13    PI = 4.0f * nPoints / (WORKER_SAMPLES_COUNT * nWorkers) ;
14    PRINTF("PI equals %f \n" , PI);
15 }
```

General Application Recovery

- How to reconstruct the lost portion of the graph?
- How to manage data versions?

CnC-OCR Applications

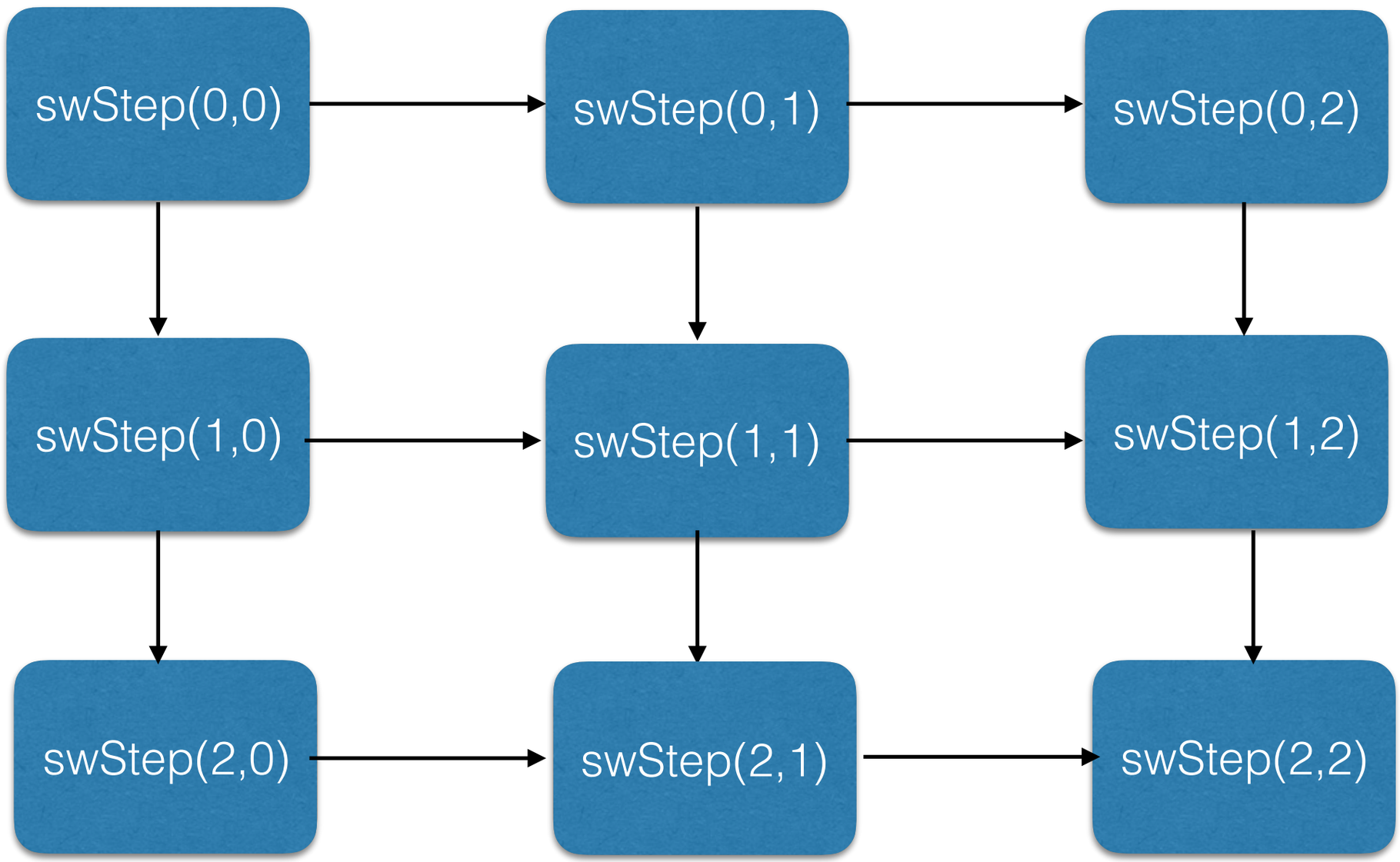
- How to reconstruct the lost portion of the graph?
  - CnC-OCR has a global knowledge of the whole graph structure
  - CnC-OCR distribution tunings provide hints about task and data locations
- How to manage data versions?
  - CnC-OCR uses single-assignment

CnC Smith-Waterman

Graph Specification

```
( swStep: i, j )
    <- [ above: i, j ]      $when(i > 0),
       [ left:  i, j ]      $when(j > 0)
    -> [ above: i+1, j ],
       [ left:  i, j+1 ],
       ( swStep: i+i, j ) $when(i+1 < #rows);
```

Smith-Waterman



CnC Smith-Waterman

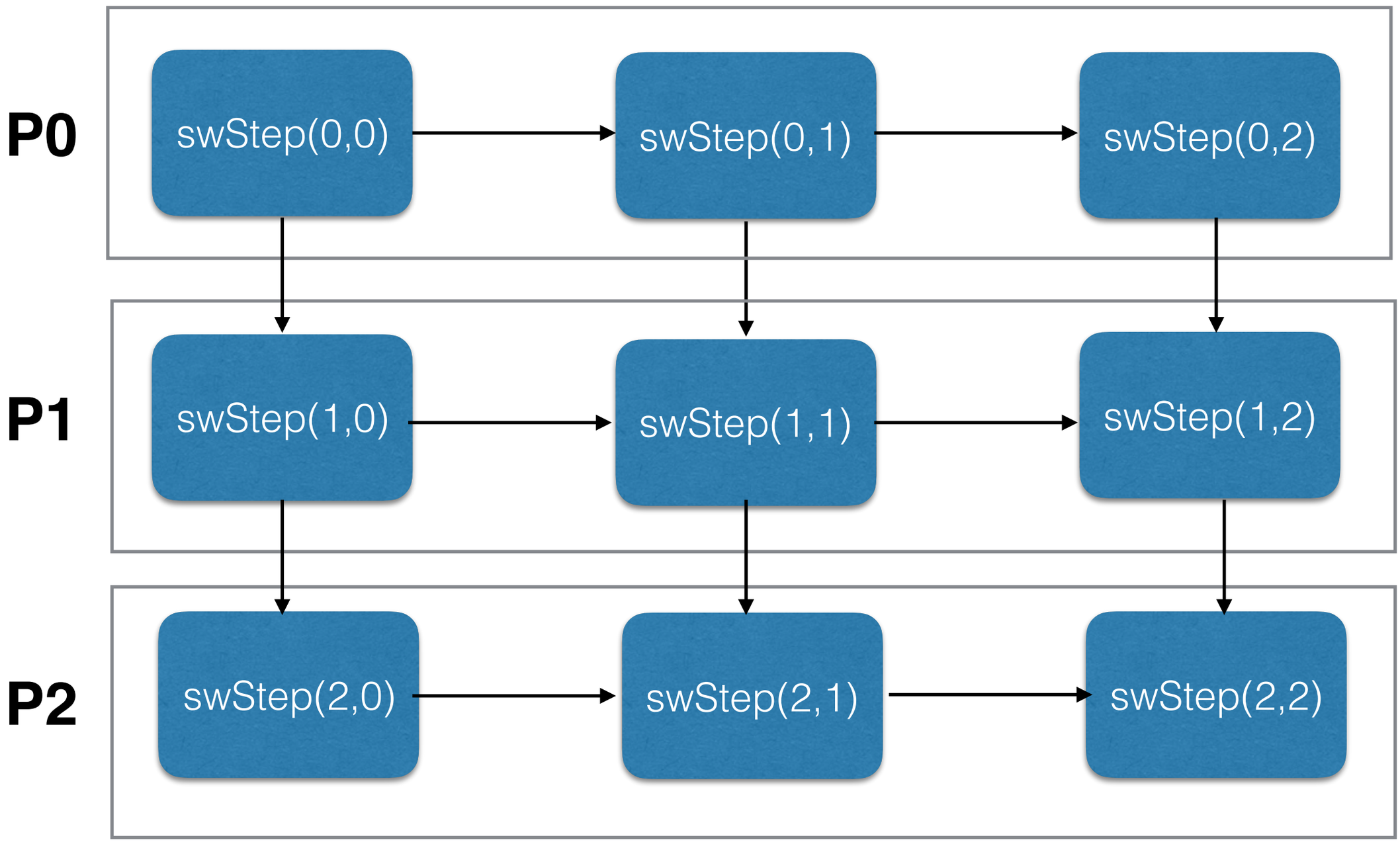
Graph Specification

```
( swStep: i, j )
    <- [ above: i, j ]      $when(i > 0),
       [ left:  i, j ]      $when(j > 0)
    -> [ above: i+1, j ],
       [ left:  i, j+1 ],
       ( swStep: i+i, j ) $when(i+1 < #rows);
```

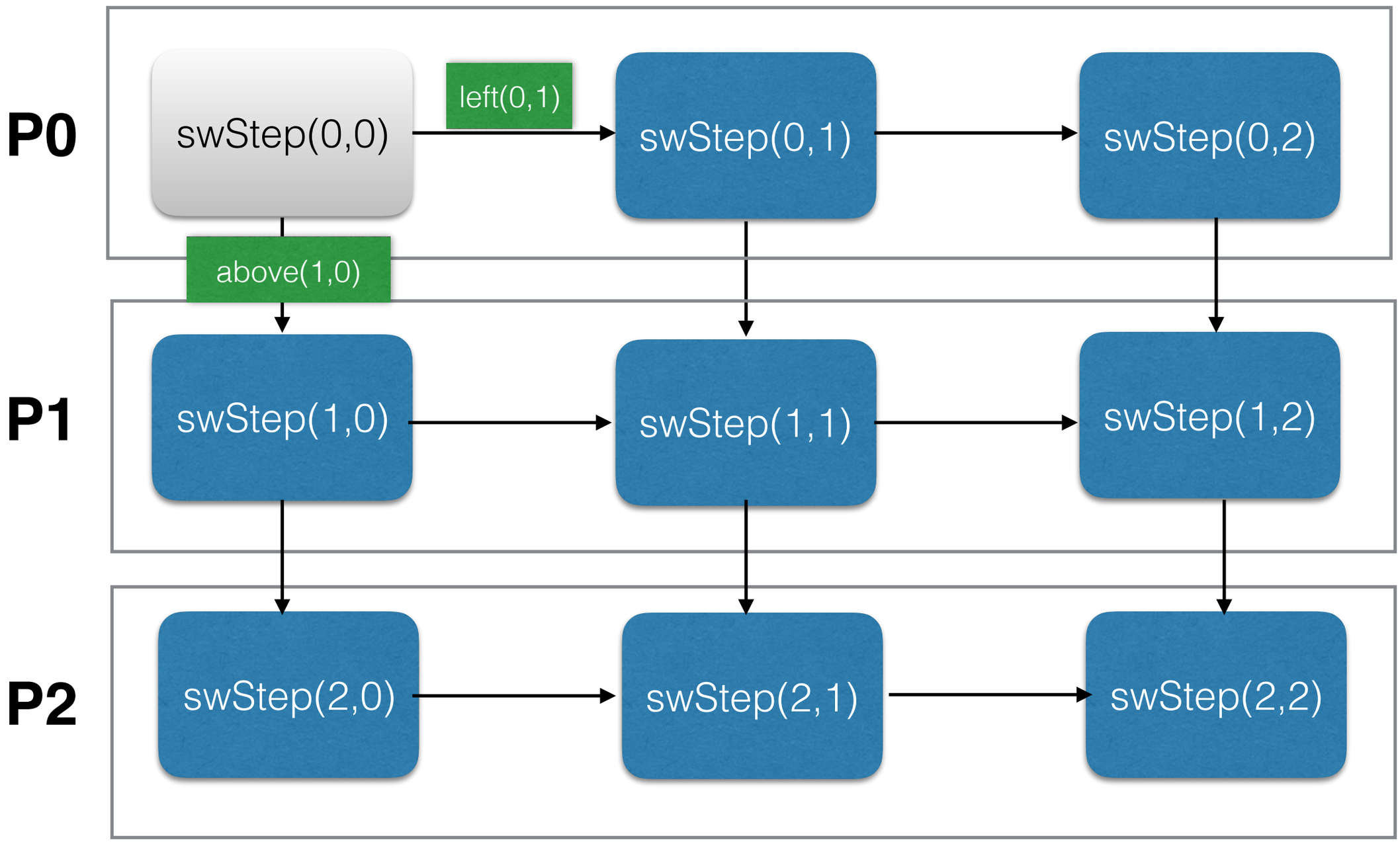
Tuning Specification (Row-Block Distribution)

```
( swStep ): { distfn: ( i ) % $RANKS };
```

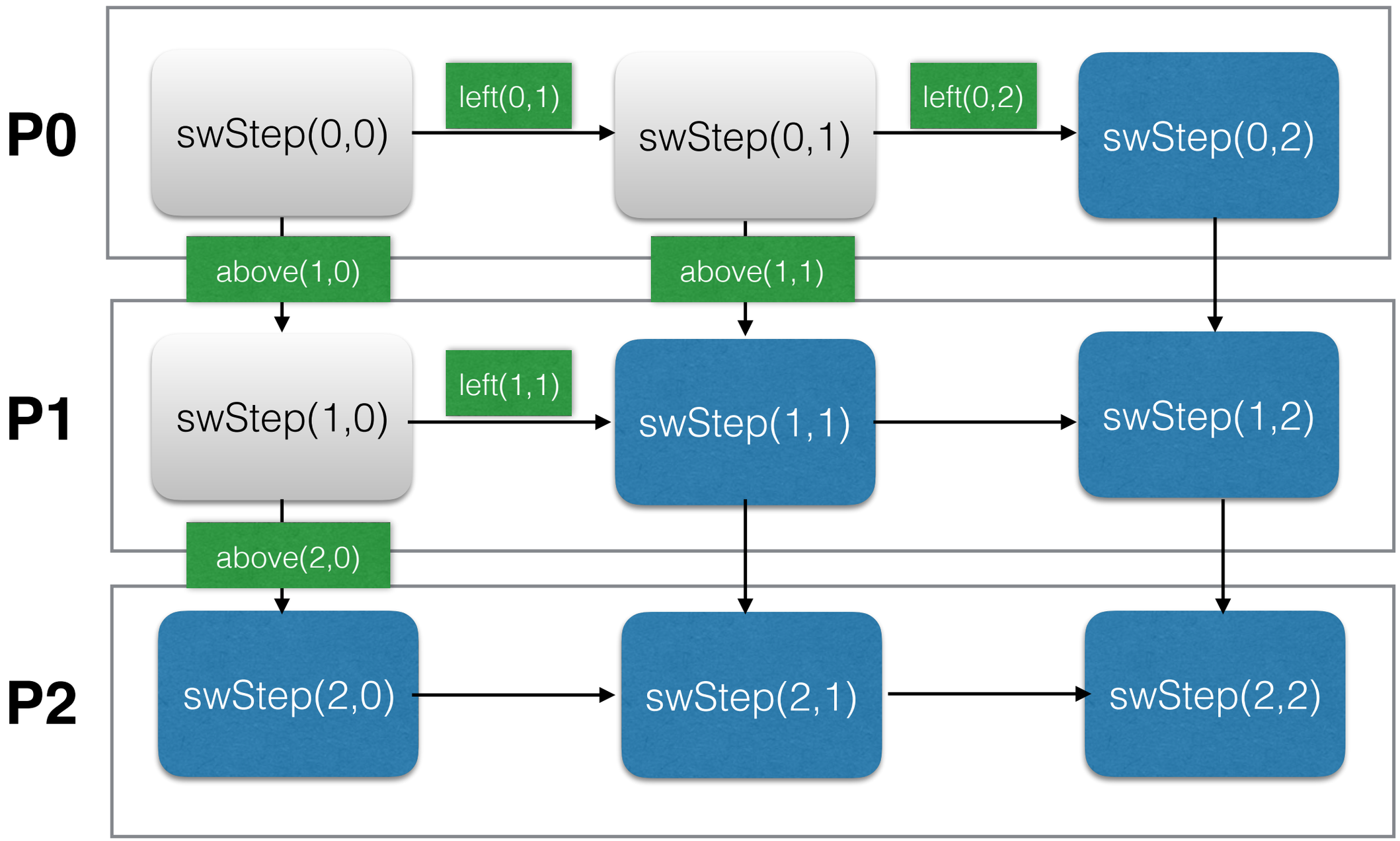
CnC Smith-Waterman



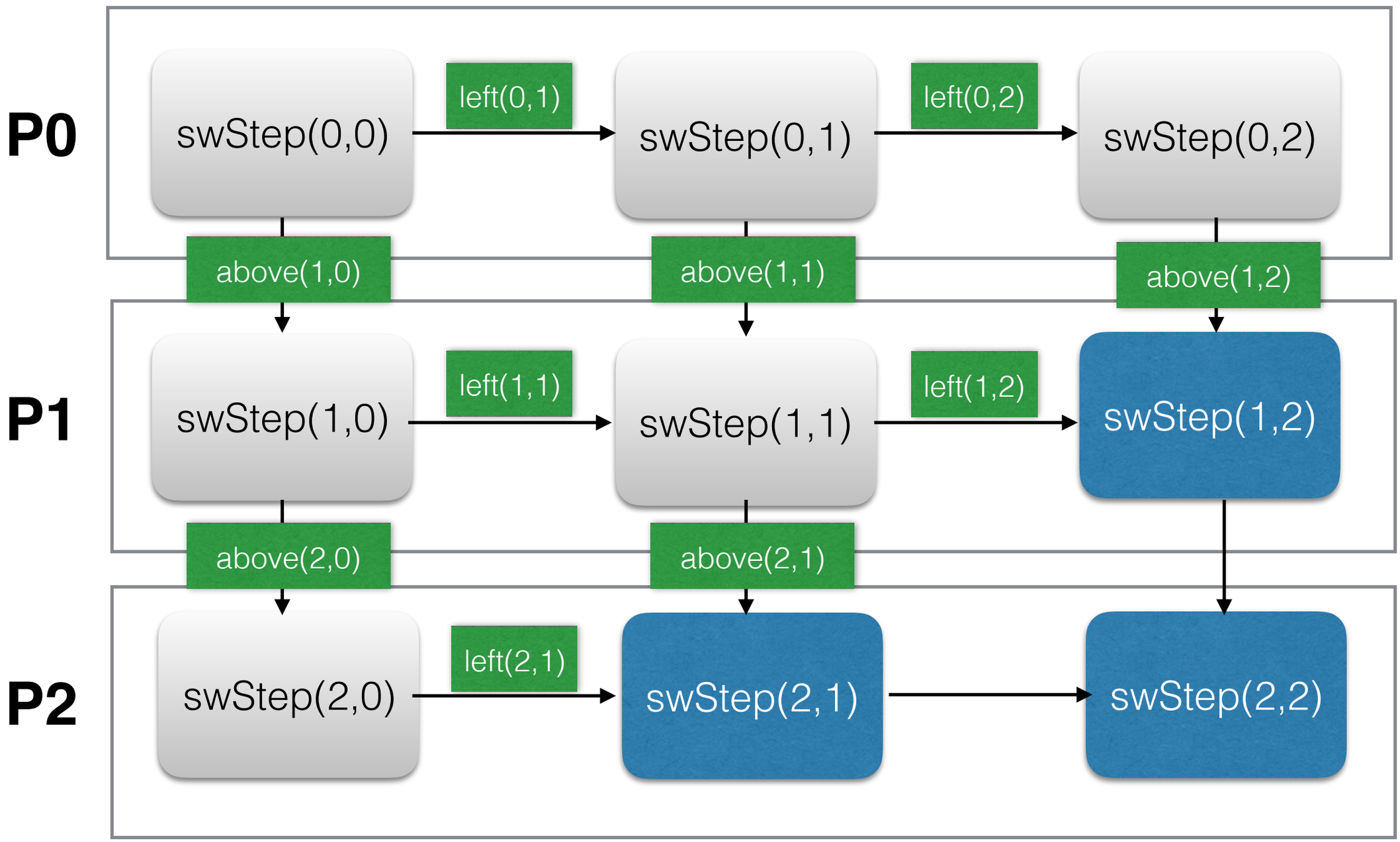
CnC Smith-Waterman



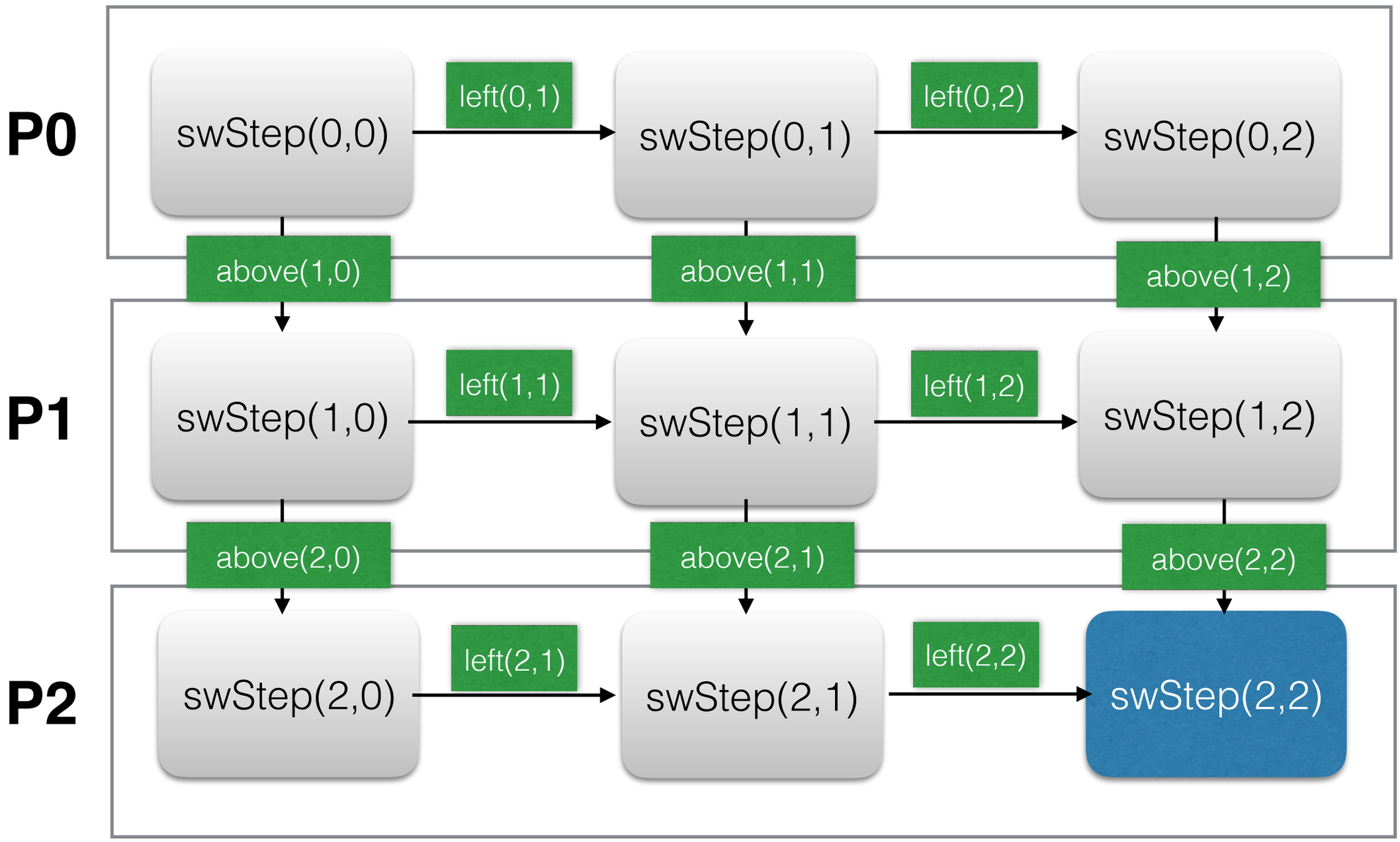
CnC Smith-Waterman



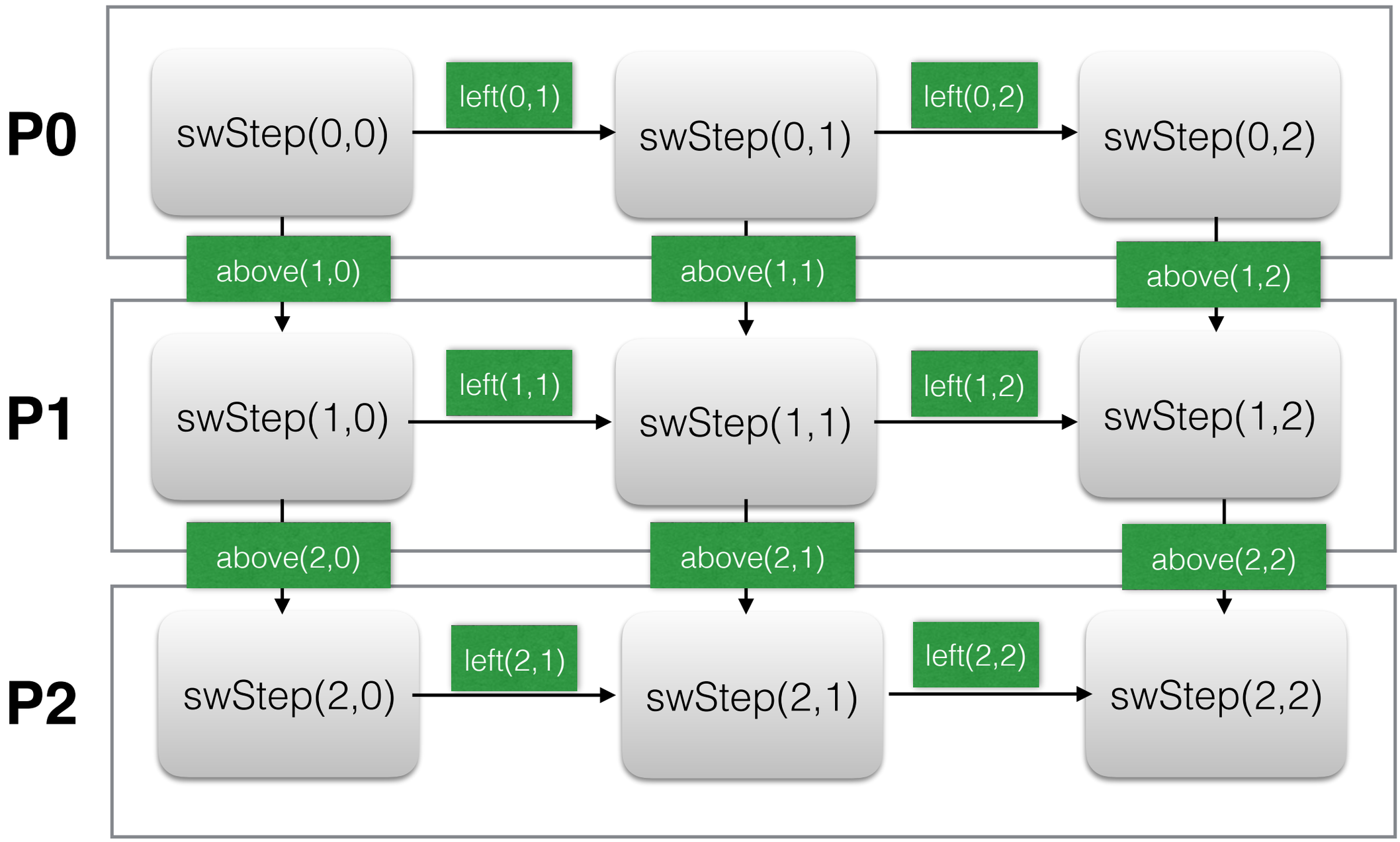
CnC Smith-Waterman



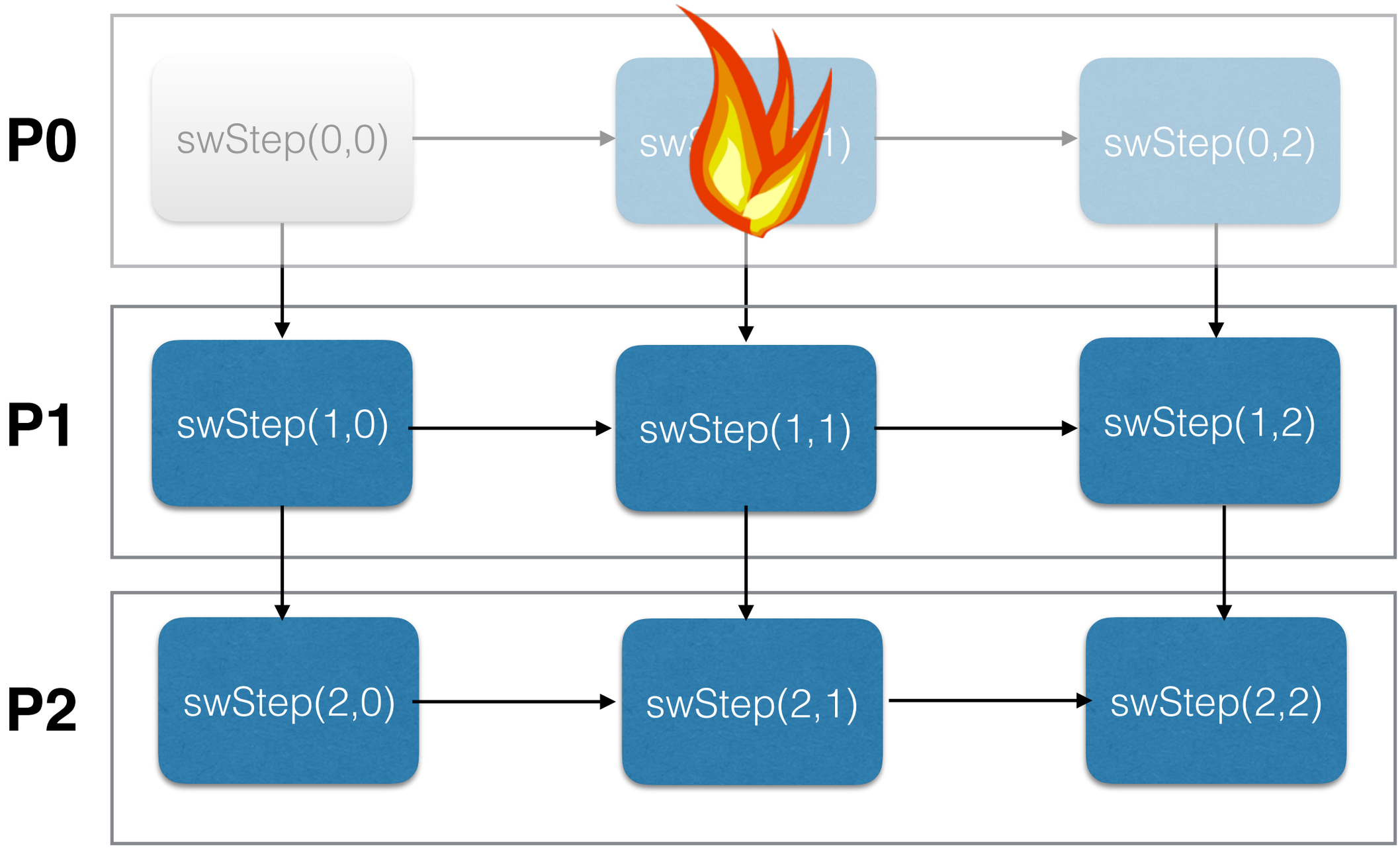
CnC Smith-Waterman



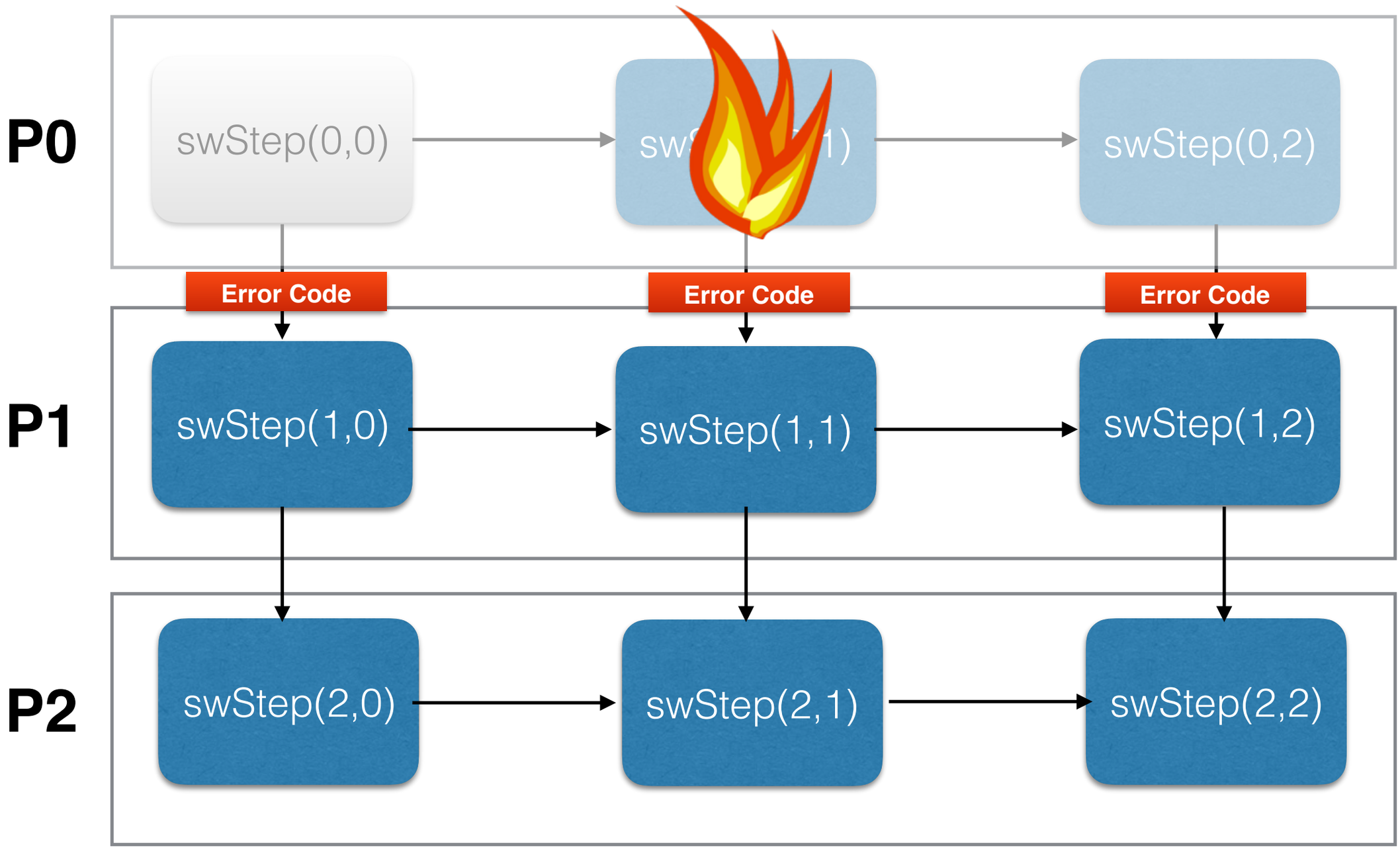
CnC Smith-Waterman



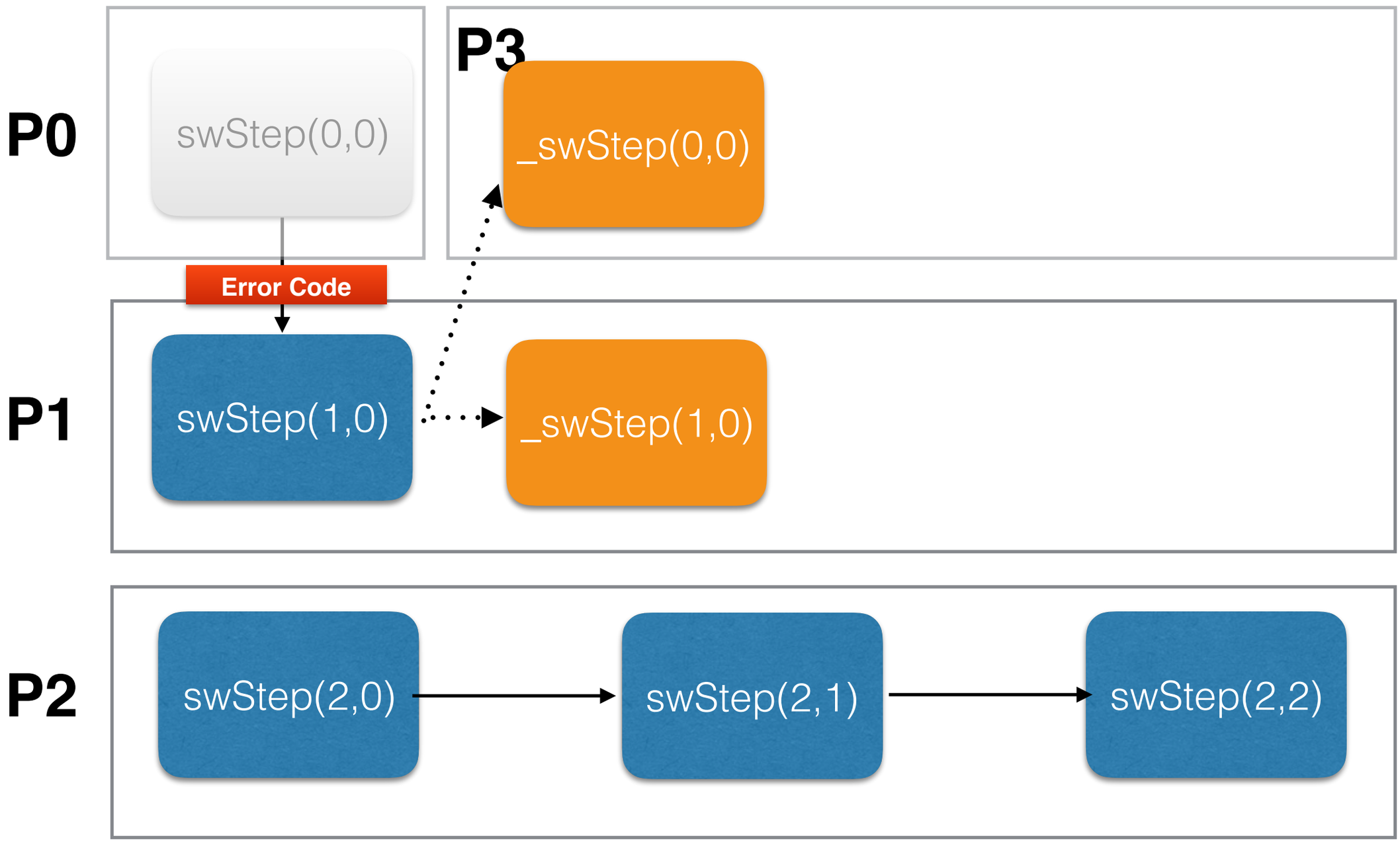
CnC Smith-Waterman



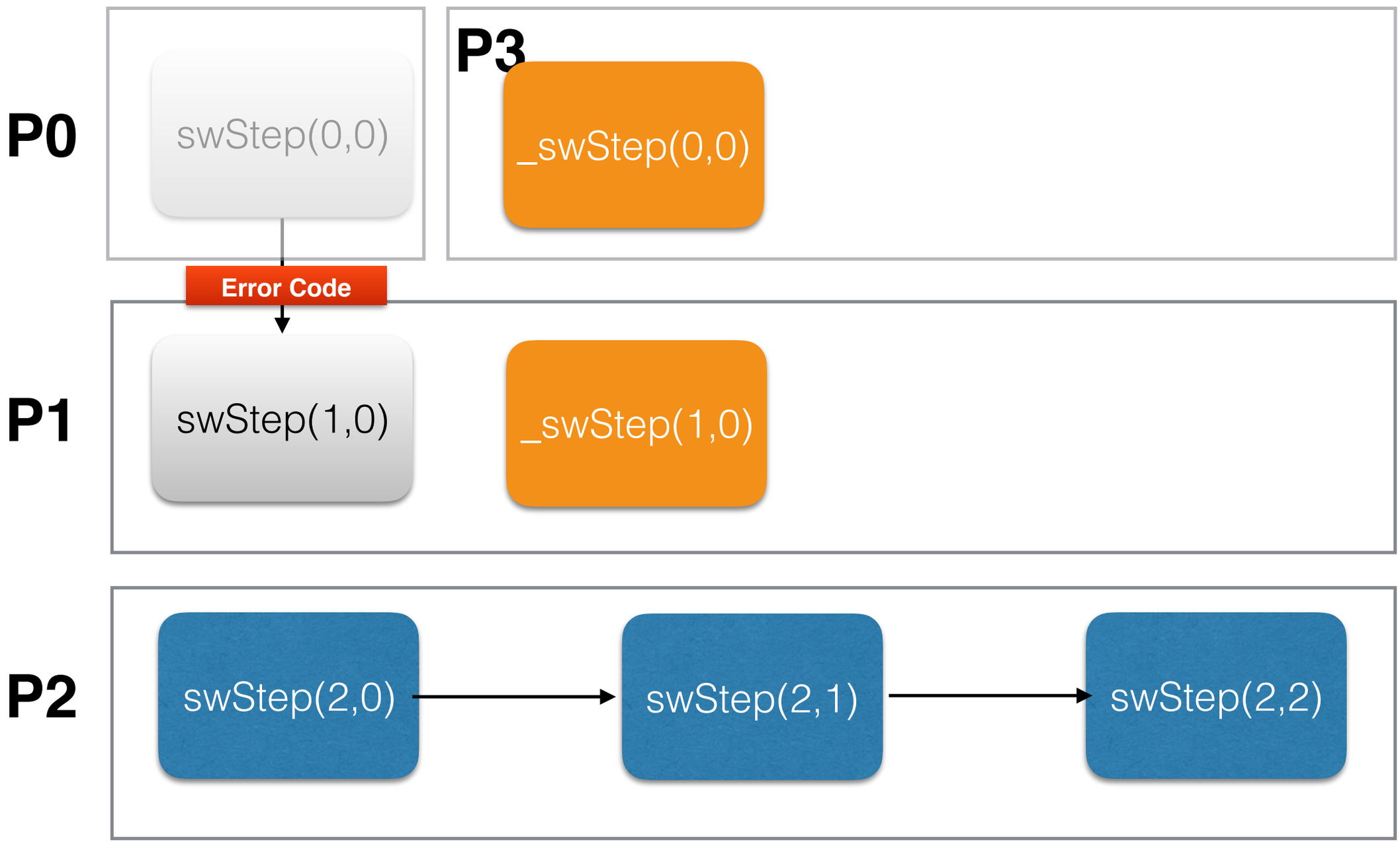
CnC Smith-Waterman



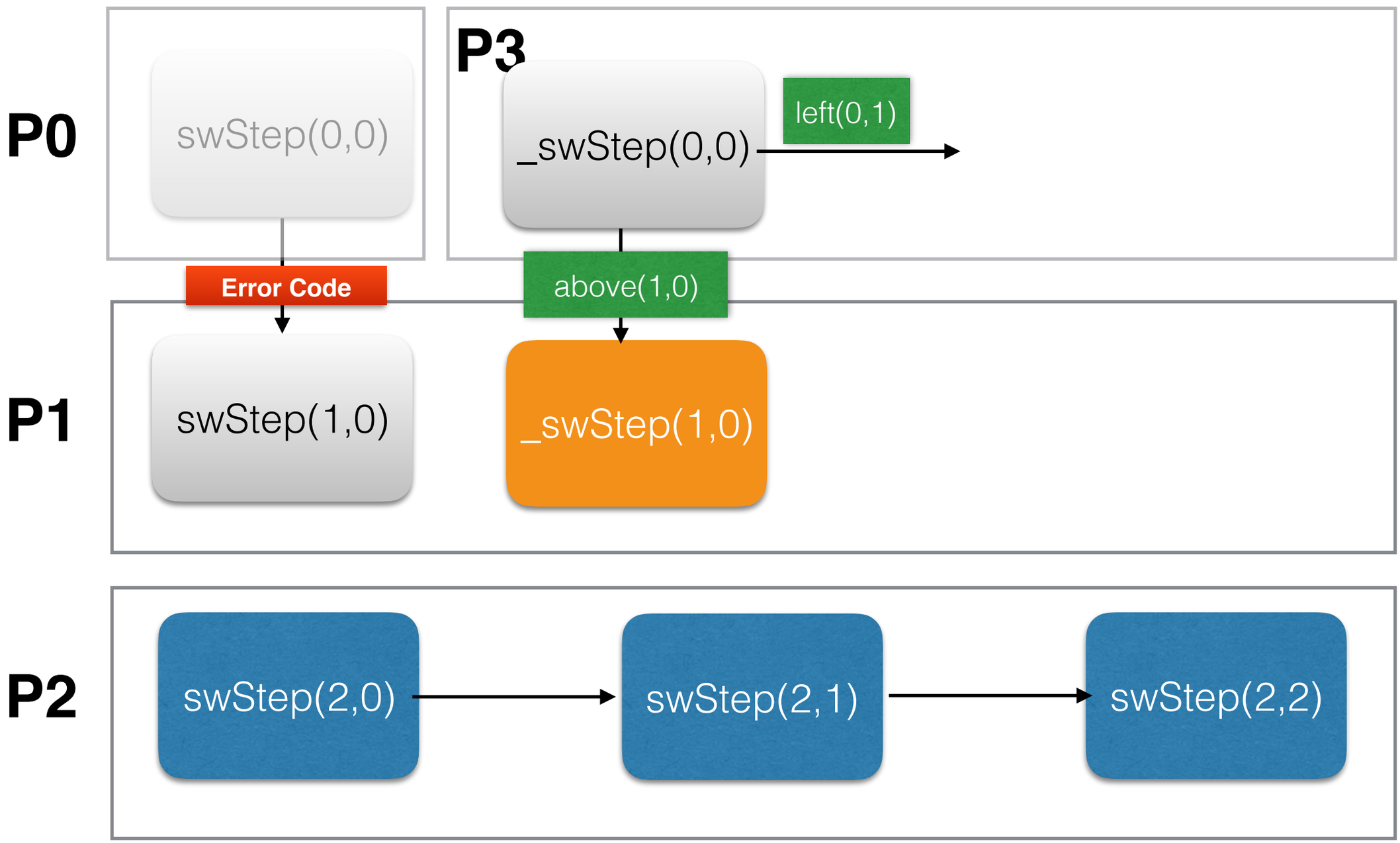
Smith-Waterman



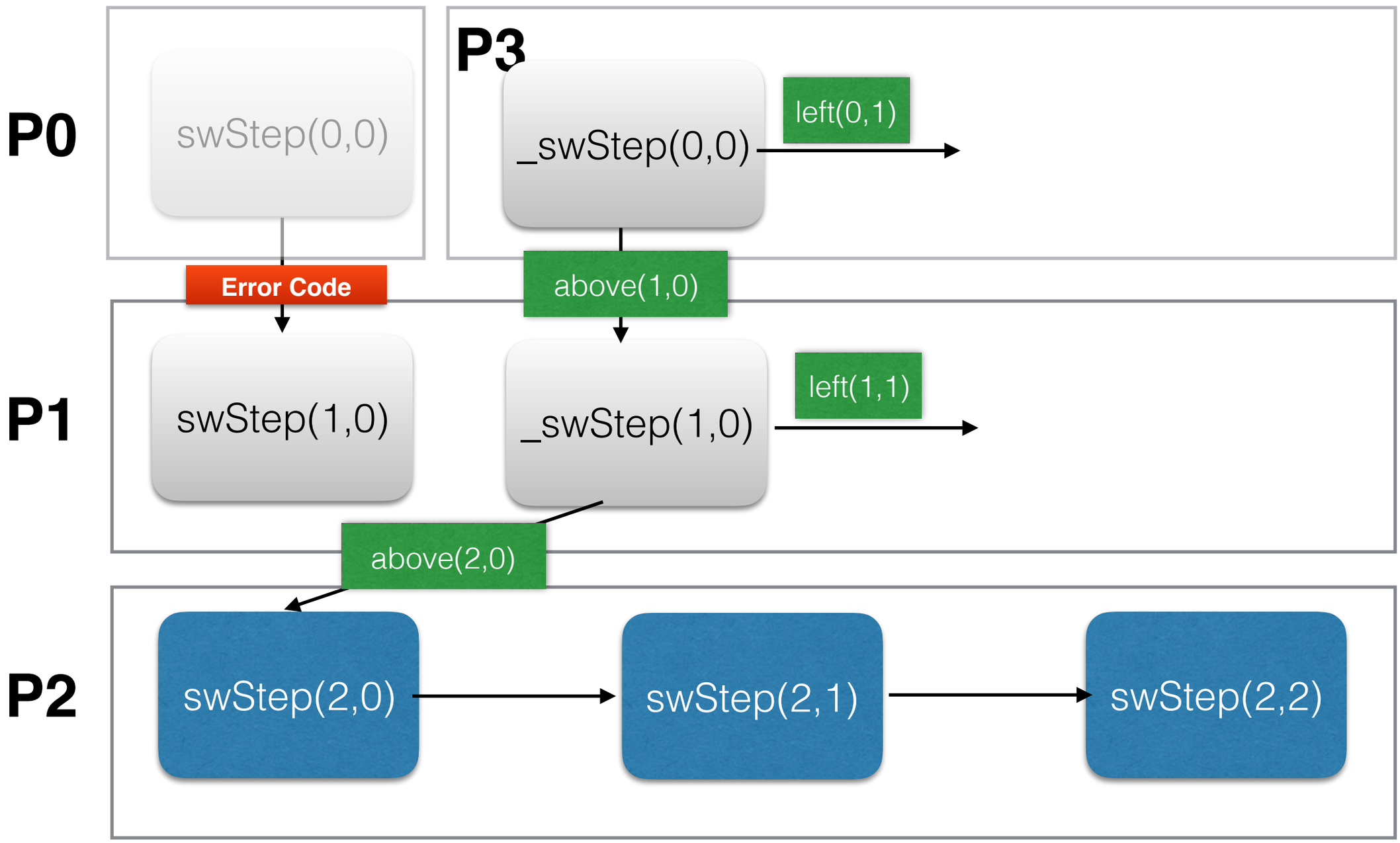
Smith-Waterman



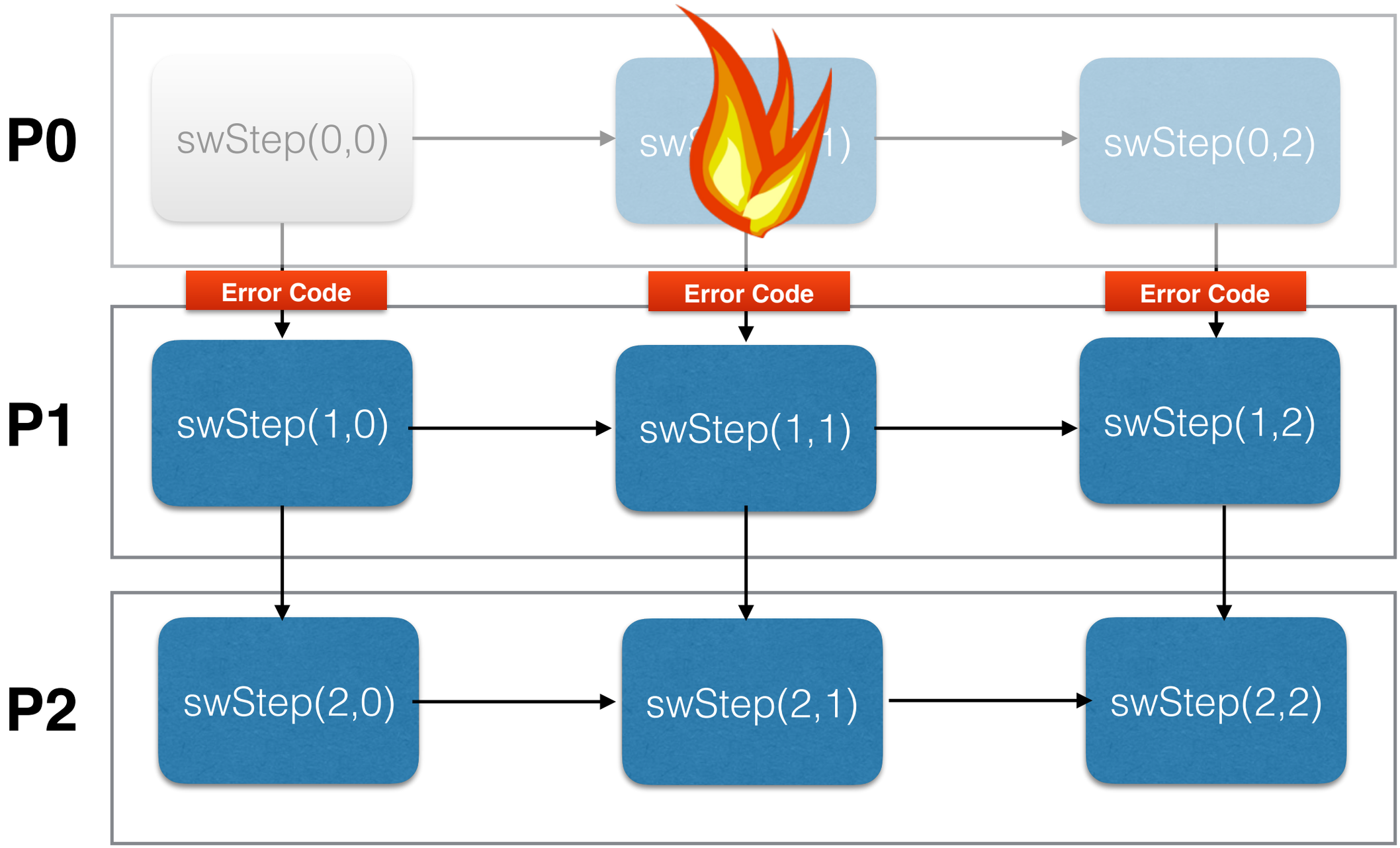
Smith-Waterman



Smith-Waterman



CnC Smith-Waterman



Conclusion

- OCR with user level fault tolerance
  - Failure detection: using MPI-ULFM
  - Failure propagation: using local proxy event
- CnC-OCR single assignment, global dependence knowledge and tunings simplify application recovery
- Future Work:
  - Performance Evaluation
  - Support a more general subset of OCR