Programmability and Performance of Parallel ECS-based Simulation of Multi-Agent Exploration Models

SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Pellegrini
Francesco Quaglia

High Performance and Dependable Computing Systems Group
Sapienza, University of Rome

PADABS 2014

Event and Cross State (ECS) Dependency

- In traditional DES:
  - interactions happen via timestamped event exchanges among LPs
  - each LP keeps a portion of the whole simulation state

Event and Cross State (ECS) Dependency

- In traditional DES:
  - interactions happen via timestamped event exchanges among LPs
  - each LP keeps a portion of the whole simulation state

- Then, this is a legal code in DES:

```
1 void *my_simulation_state = malloc(SIZE);
2 memcpy(my_simulation_state, my_content, SIZE);
3 void *evt_payload = my_simulation_state;
4 ScheduleEvent(target, timestamp, EVENT_TYPE, evt_payload, SIZE);
```

Event and Cross State (ECS) Dependency

- In traditional DES:
  - interactions happen via timestamped event exchanges among LPs
  - each LP keeps a portion of the whole simulation state

- Then, this is a legal code in DES:

```
1 void *my_simulation_state = malloc(SIZE);
2 memcpy(my_simulation_state, my_content, SIZE);
3 void *evt_payload = my_simulation_state;
4 ScheduleEvent(target, timestamp, EVENT_TYPE, evt_payload, SIZE);
```

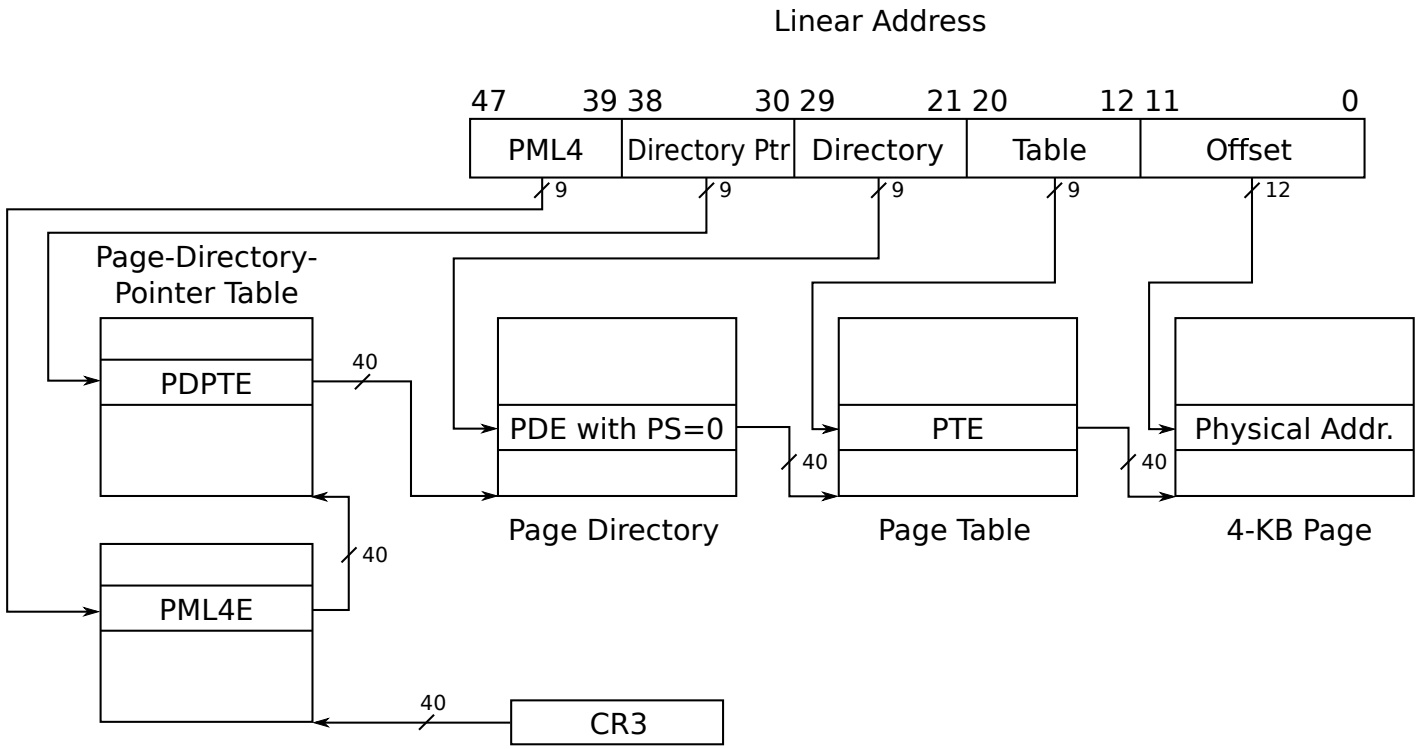
- In sequential DES simulation, so far so good.
- What if this model is executed in a Parallel DES environment?
  - Think of Optimistic Synchronization!

Step 1: Materializing Cross-State Dependencies

- To *transparently* detect accesses to other LPs' states we rely on an `x86_64` kernel-level memory management architecture

Step 1: Materializing Cross-State Dependencies

- To *transparently* detect accesses to other LPs' states we rely on an `x86_64` kernel-level memory management architecture

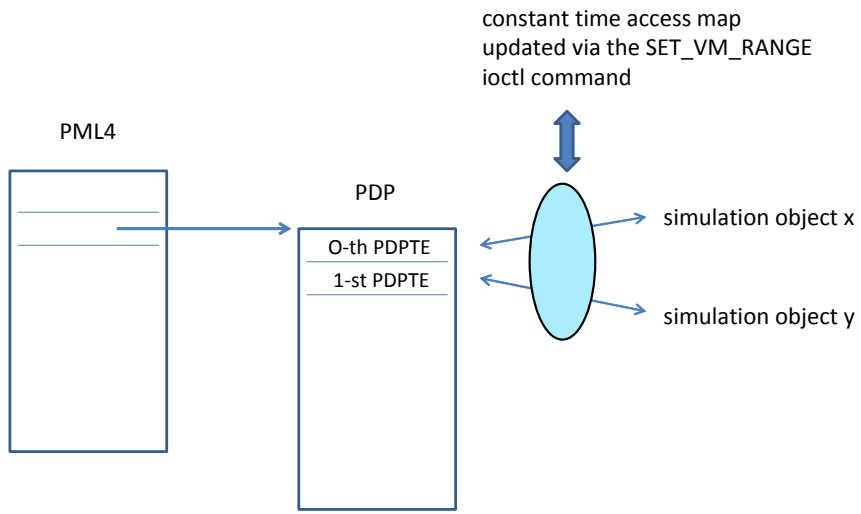


Memory Allocation Policy

- LPs use virtual memory according to *stocks*
- Memory requests are intercepted via malloc wrappers (DyMeLoR)
- Upon the first request, an interval of page-aligned virtual memory addresses is reserved via `mmap` POSIX API (a *stock*)
- This is a set of empty-zero pages: a null byte is written to make the kernel actually allocate the chain of page tables
- One stock gives 1GB of available memory to each LP

Memory Access Management

- 99% handled via a Linux Kernel Module
- A LKM creates a device file accessible via `ioctl`
- `SET_VM_RANGE` command associates stocks with LPs
- A kernel-level map (accessible in constant time) is created:
  - Each stock is logically related to one entry of a PDP page-table
  - The id of the LP who the stock belongs to is registered



Memory Access Management

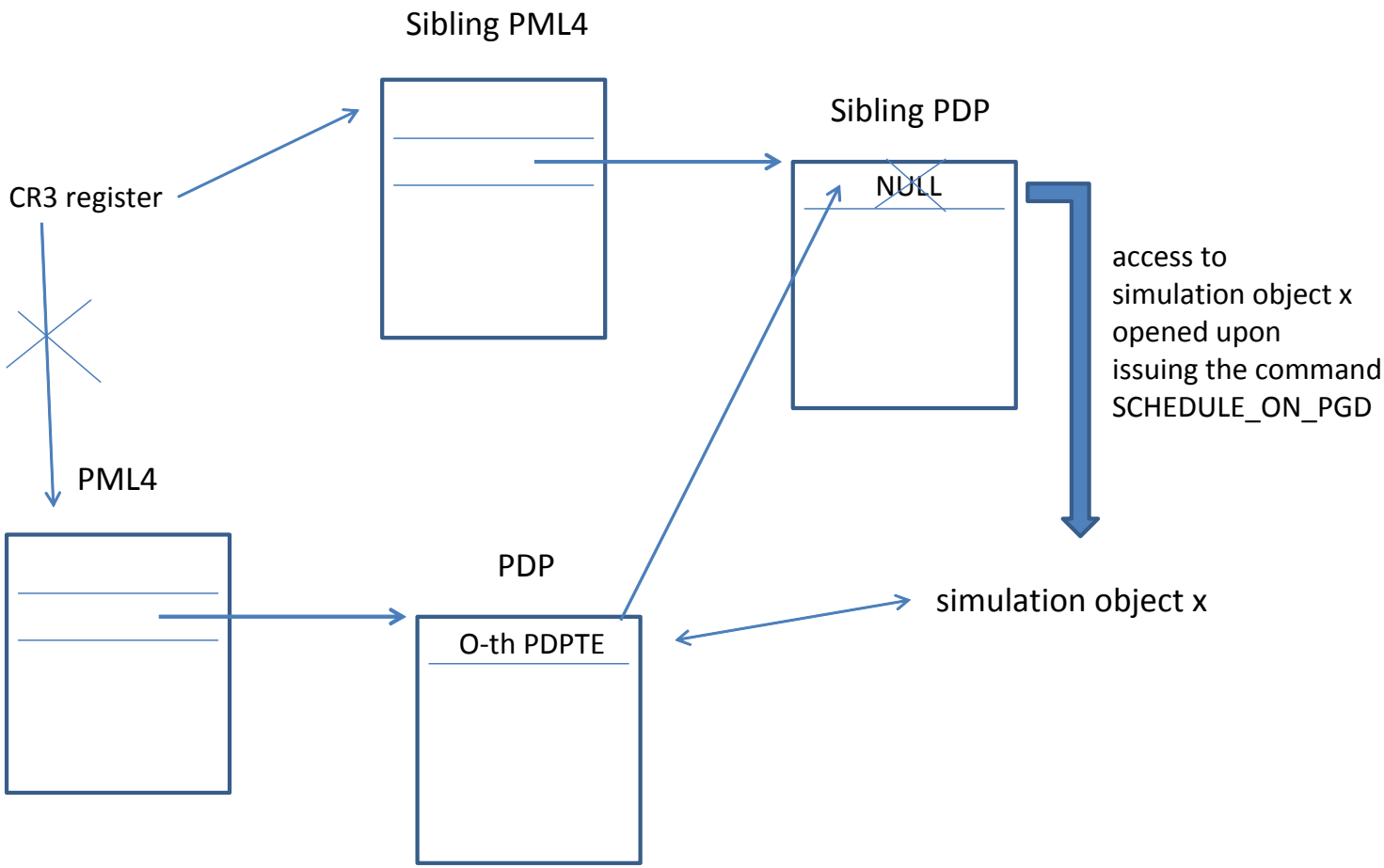
- When LP j accesses LP i's state, we could know that by the memory address
- We target SMP Simulation: memory protection is not an option
- Every worker thread is associated with a sibling PML4 entry:
  - They point same PDP entries...
  - ...but with different privileges!

Memory Access Management

- When LP j accesses LP i's state, we could know that by the memory address
- We target SMP Simulation: memory protection is not an option
- Every worker thread is associated with a sibling PML4 entry:
  - They point same PDP entries...
  - ...but with different privileges!

- The `SCHEDULE_ON_PGD` command brings the execution in *simulation-object mode*:
  - The only accessible stock is dispatched LP's one
  - This operation leads to a change in the `CR3` hardware register

Memory Access Management



Cross-State Dependency Materialization

- If other LPs' stocks are accessed, we have a memory fault
- This is the materialization of a Cross-State Dependency

- Yet, this page fault cannot be traditionally handled:
  - Memory has already be validated via `mmap` at simulation startup
  - The Linux kernel would simply reallocate new pages
  - For the same virtual page we would have multiple page table entries!

Step 2: Event and Cross-State Synchronization (ECS)

- At startup we change the IDT table to redirect the page-fault handler pointer to a specific ECS handler
- Upon a real segfault, the original handler is called
- Otherwise, the ECS handler pushes control back to user mode:
  - Execution goes back into *platform mode*
  - `CR3` is switched back to the original PML4 table
  - The simulation kernel can access any memory buffer required for supporting synchronization

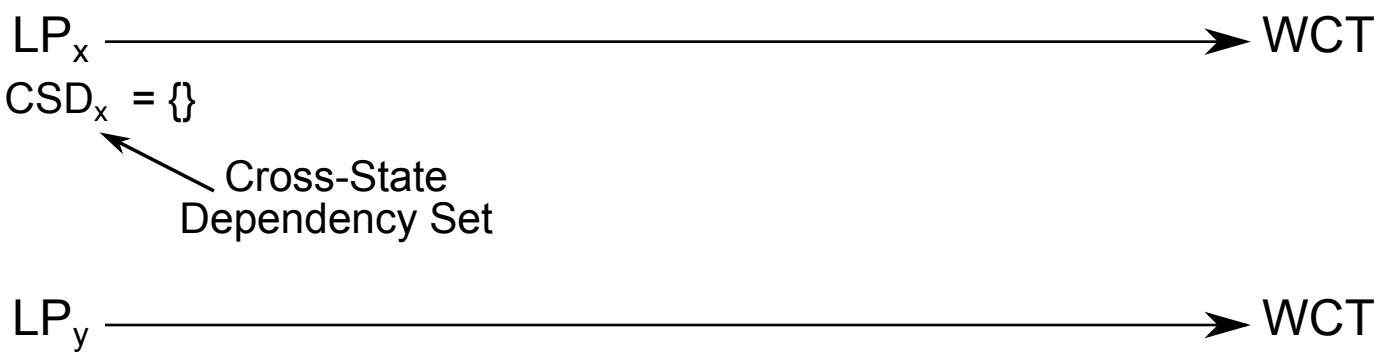
ECS System

Property

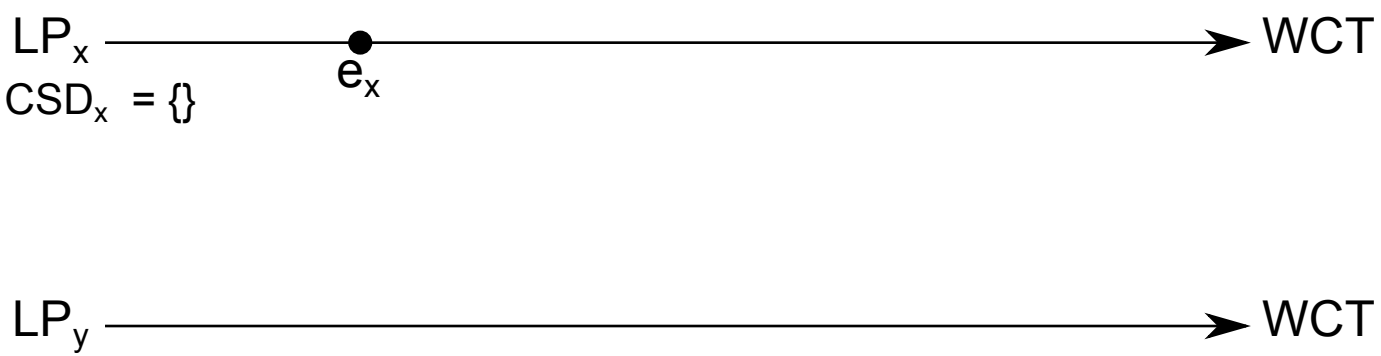
When a Cross-State Dependency is materialized at simulation time T, the involved LP observes the state snapshot that would have been observed in a sequential-run.

- To enforce this we introduce:
  - temporary LP blocking: the execution of an event can be suspended
  - *rendez-vous events*: system-level simulation events not causing state updates

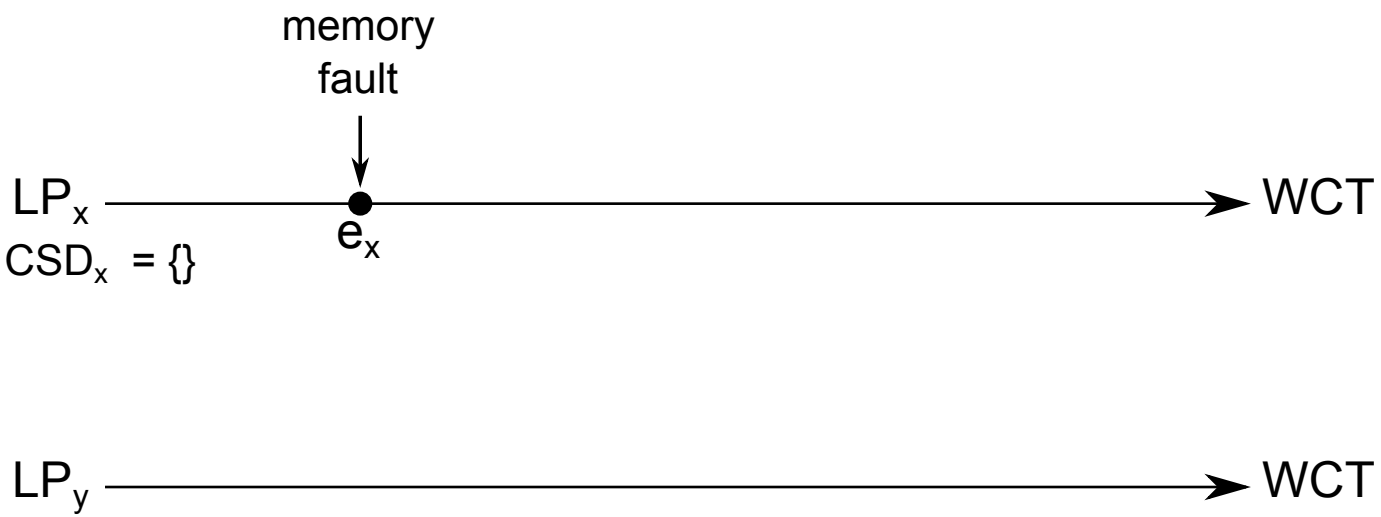
ECS System



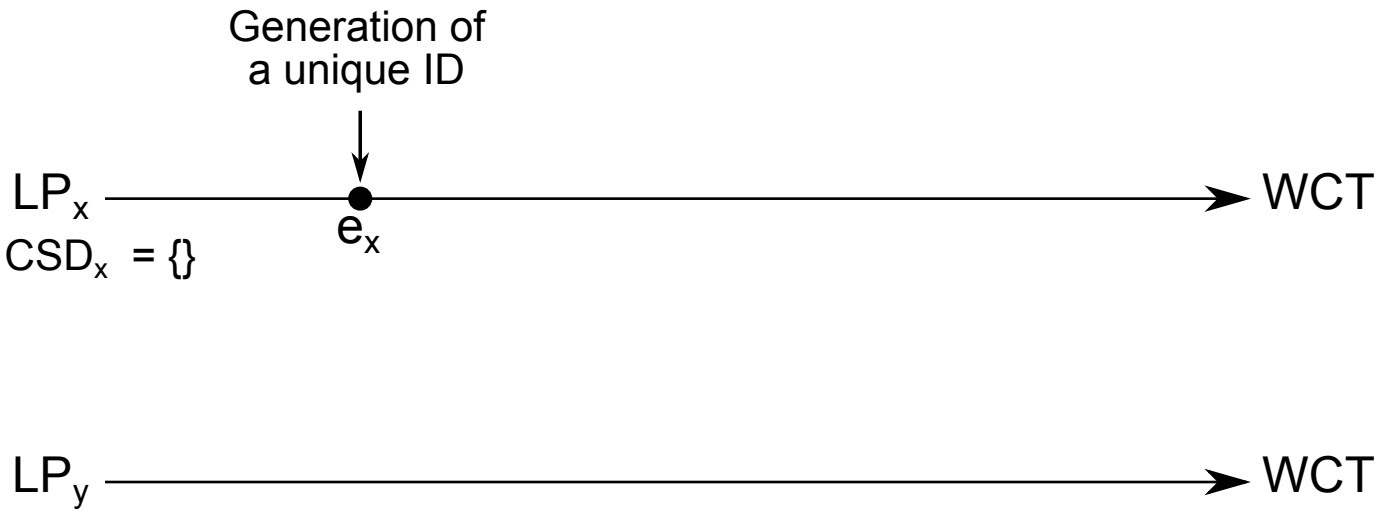
ECS System



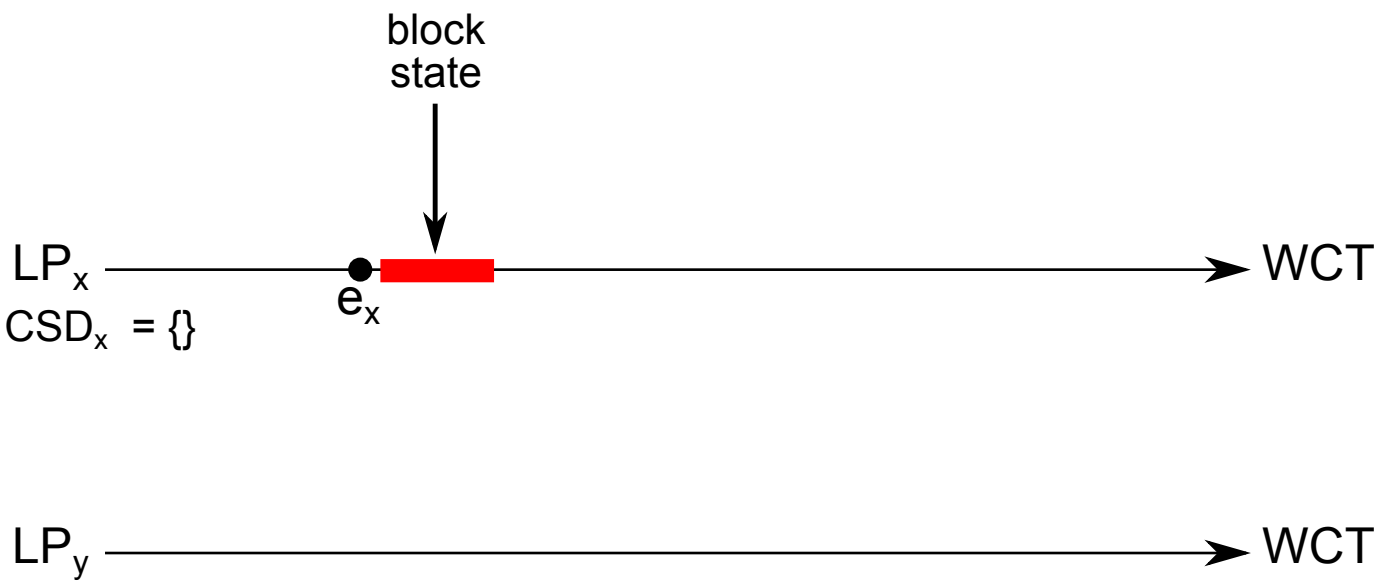
ECS System



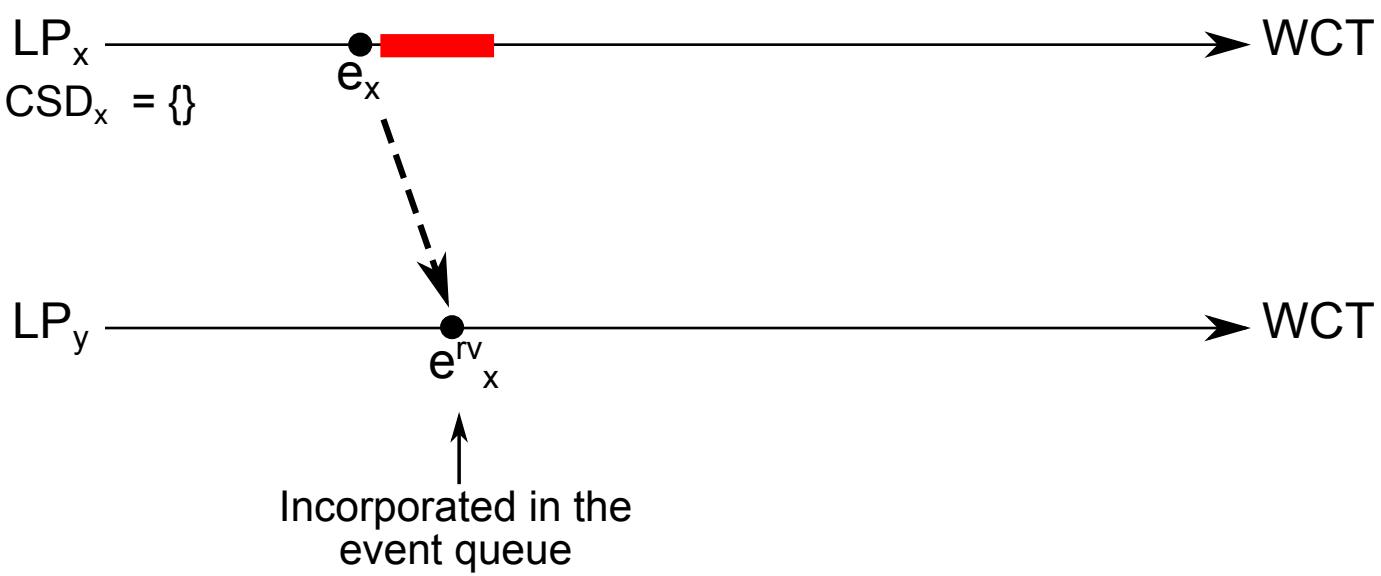
ECS System



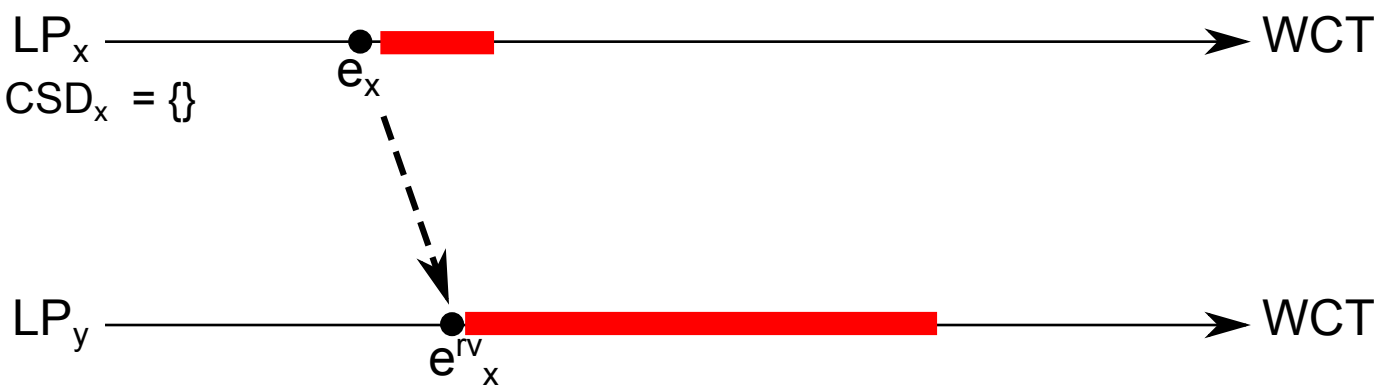
ECS System



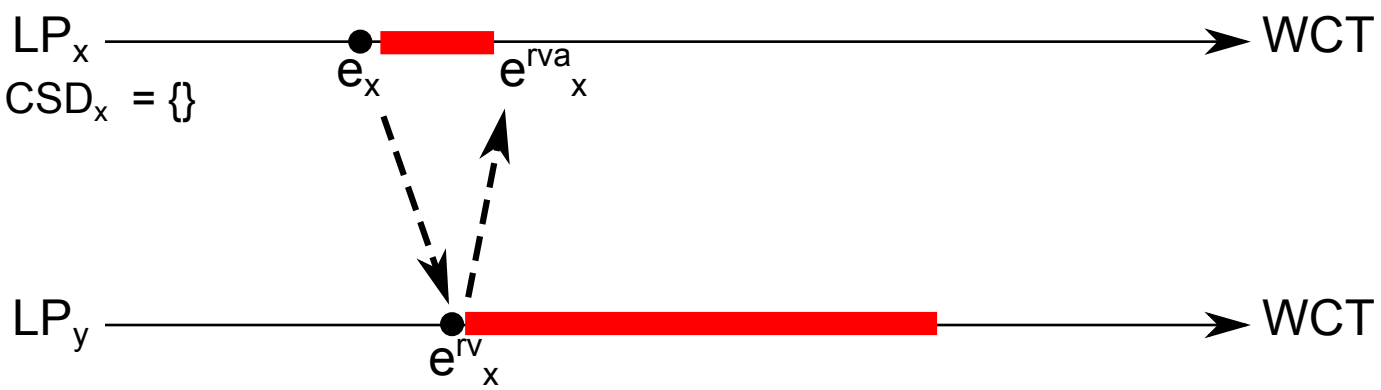
ECS System



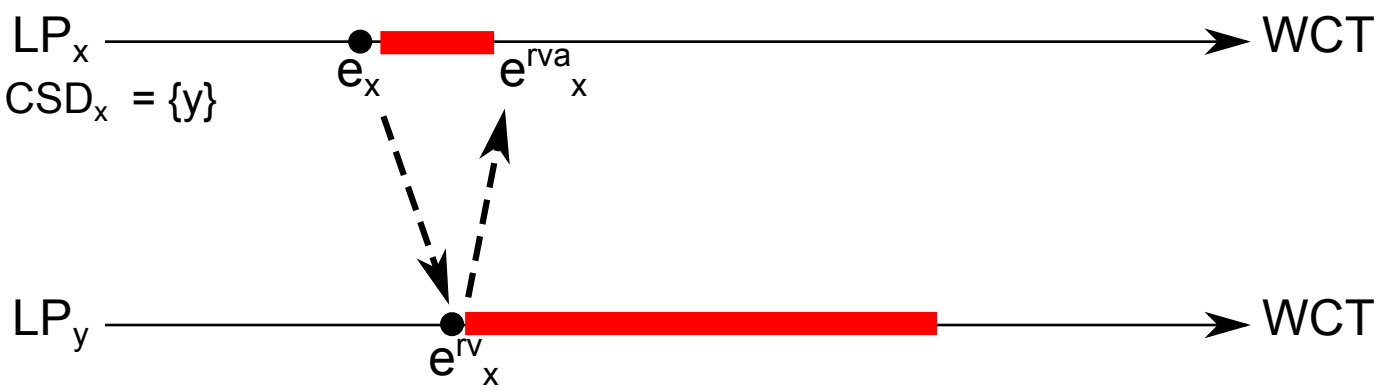
ECS System



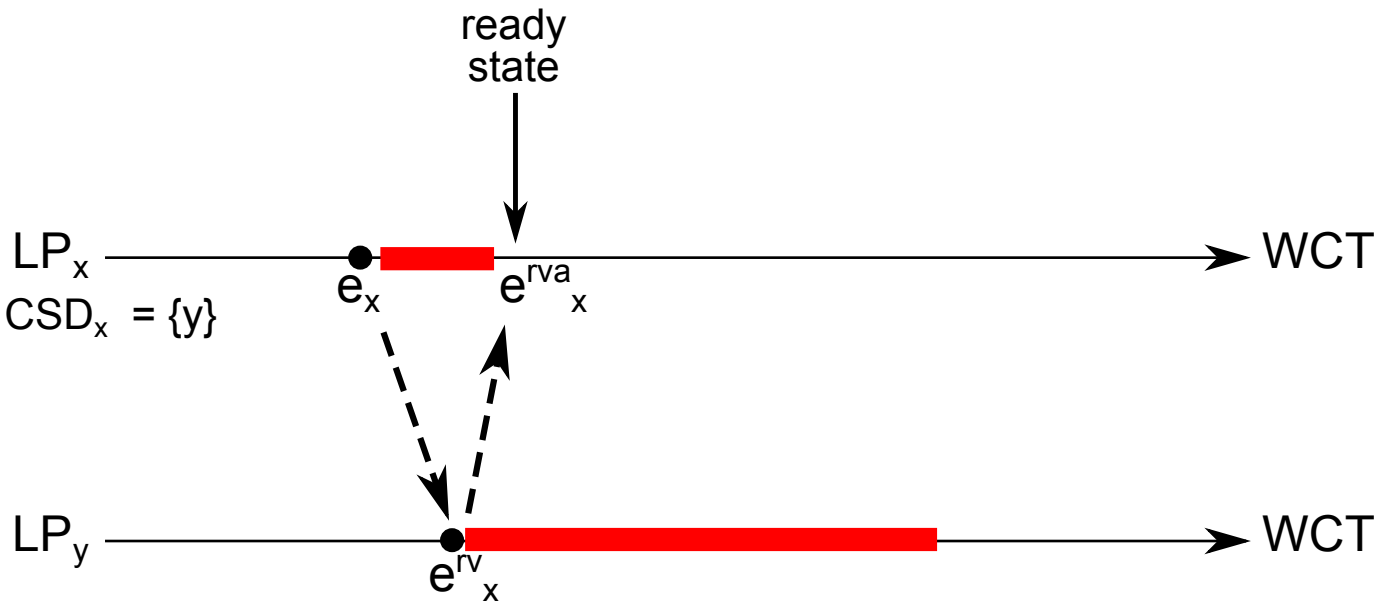
ECS System



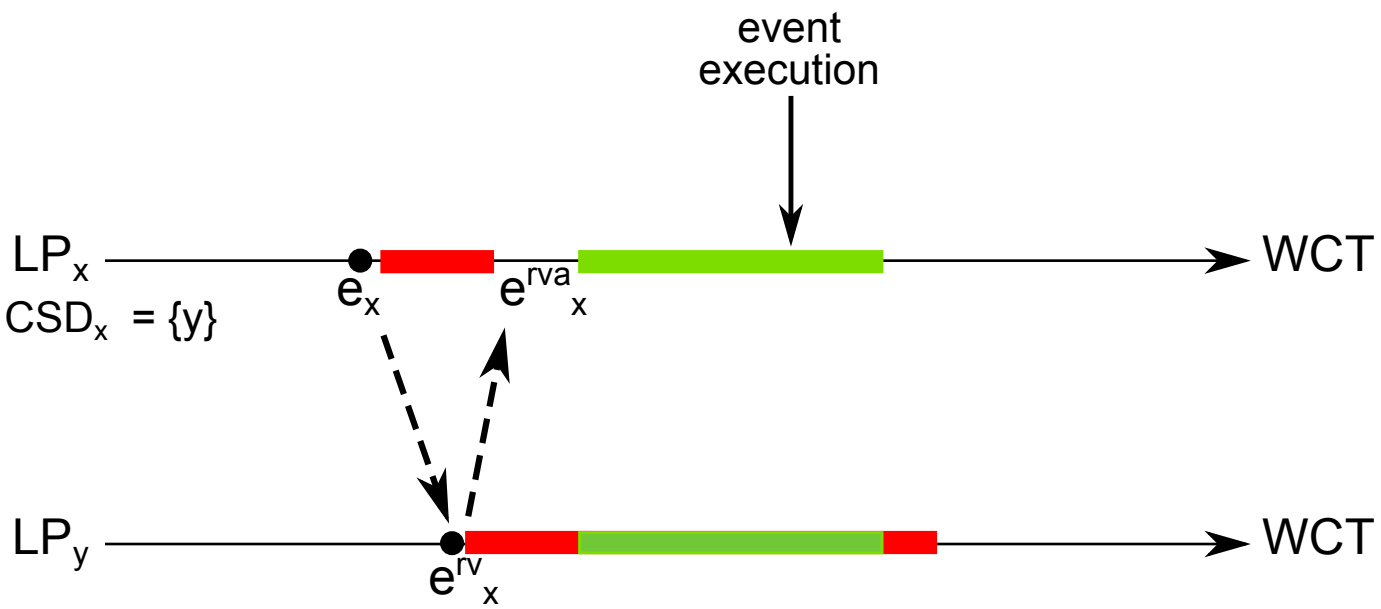
ECS System



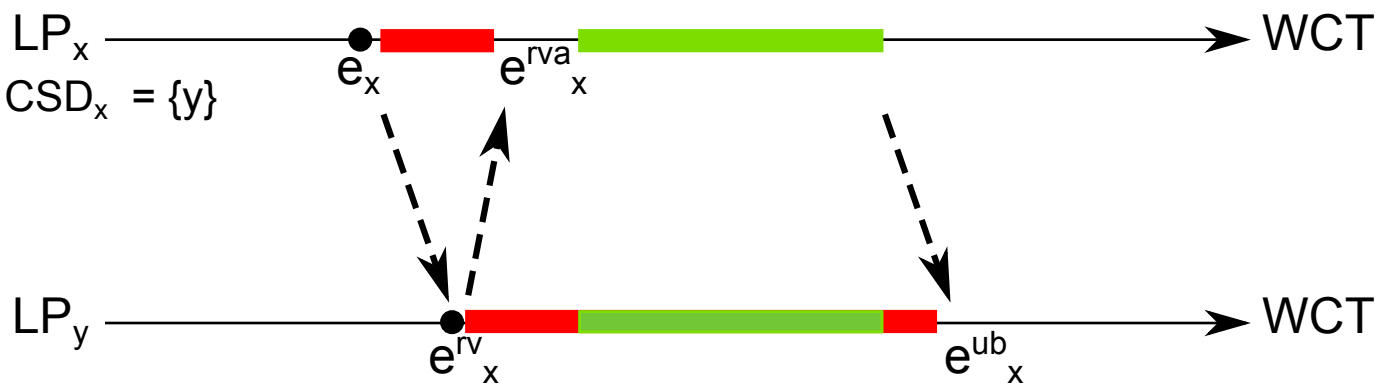
ECS System



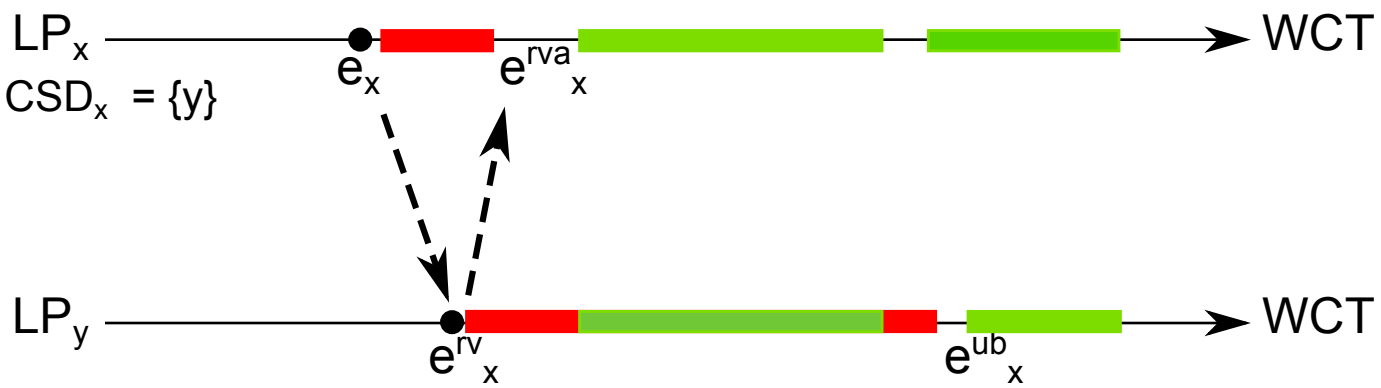
ECS System



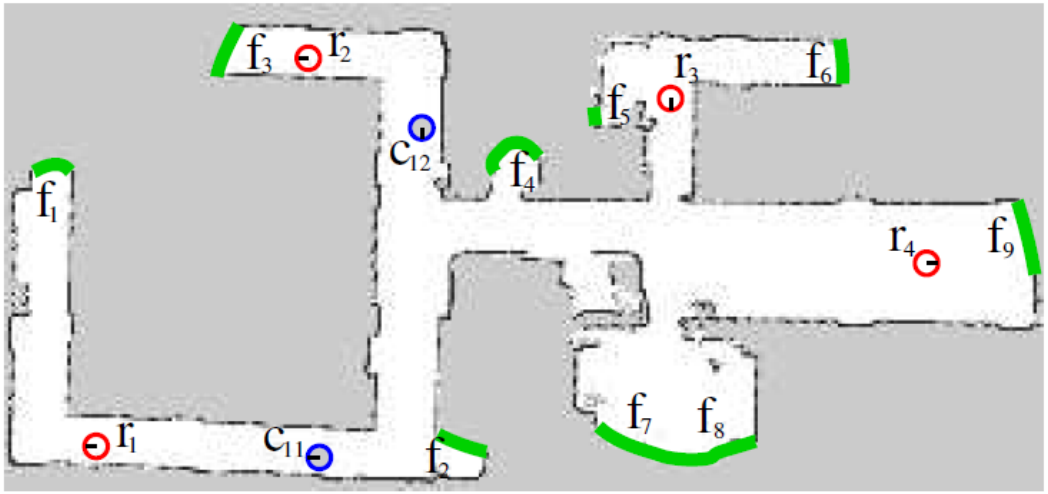
ECS System



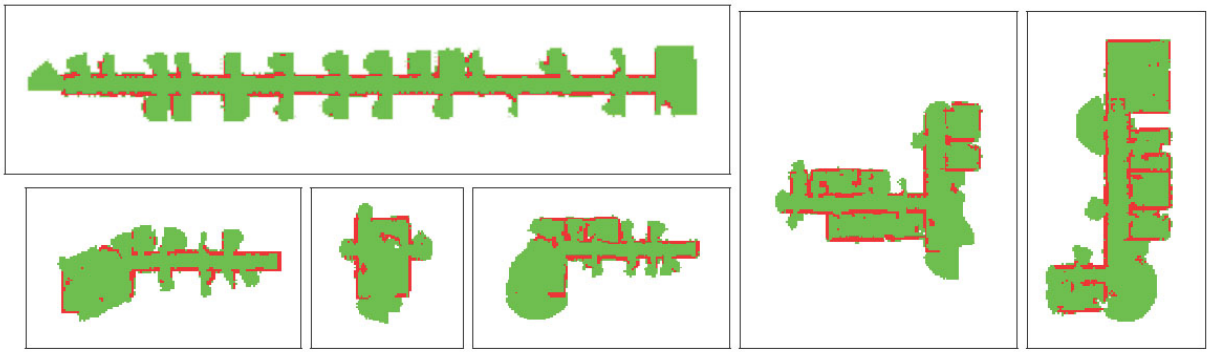
ECS System



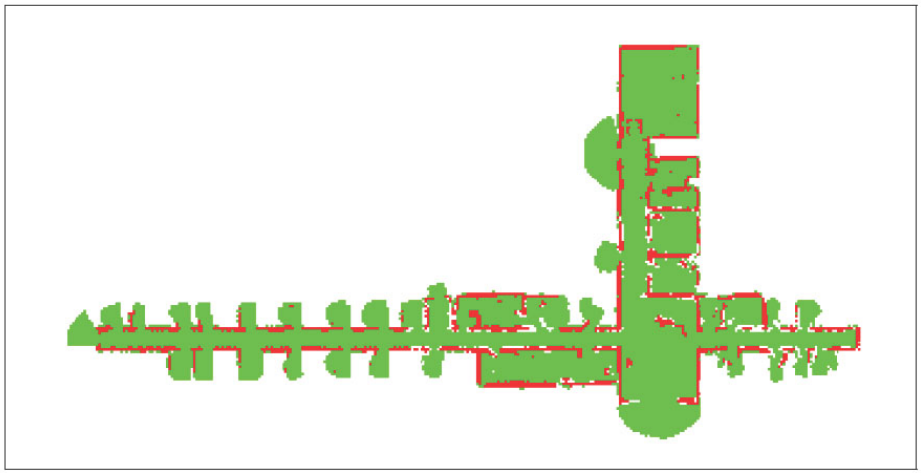
Programmability and Performance Study: Distributed Multi-Robot Exploration and Mapping



Programmability and Performance Study: Distributed Multi-Robot Exploration and Mapping



(a) input maps



(b) merged map

Programmability and Performance Study: Distributed Multi-Robot Exploration and Mapping

- The map is constructed online
- Robots explore independently, until they accidentally meet:
  1. they use their sensors to estimate their mutual physical position
  2. they create a rendez-vous point to verify the estimation's goodness
  3. if the hypothesis is verified, they exchange the so-far acquired data
  4. they form a cluster
- Clusters allow to explore collaboratively:
  - jointly define the next targets (reduce mapping time)
  - make a guess on the position of other robots (enlarge the cluster)

PDES Implementation Problems

- Discovering the presence of a nearby robots (number of exchanged messages)
- Estimating the respective position of the agents (many messages with the same timestamp)
- Exchanging data map information (non-negligible size)

- To solve these problems, the modeler must reason about optimistic synchronization and LPs' state separation: **no trasparency**

Using ECS: Initialization of Cells

```
1  // Allocated state
2  state = malloc(sizeof(agent_state_type));
3
4  <initialize the map>
5
6  // Allocate the presence bitmap
7  state->agents = malloc(BITMAP_SIZE(num_agents));
8  bzero(state->agents, BITMAP_SIZE(num_agents));
9  // Register the state
10 states[me] = state;
```

Using ECS: Entering a Cell

```
1 state->current_cell = event_content->cell;
2
3 // Register the position of the robot in the cell
4 cell = (cell_state_type *)states[state->current_cell];
5 cell->present_agents++;
6 SET_BIT(cell->agents, me - num_cells);
```

Using ECS: Explore the Cell

```
1 // Mark the cell as explored and "discover" the surroundings
2 state->visit_map[state->current_cell].visited = true;
3 memcpy(&state->visit_map[state->current_cell].data, cell->data,
      sizeof(unsigned int) * 6);
4
5 // Is there any other robot in the cell?
6 if(cell->present_agents > 1) {
7     <scan the bitmap>
8 }
```

Using ECS: Exchange Data with Robots

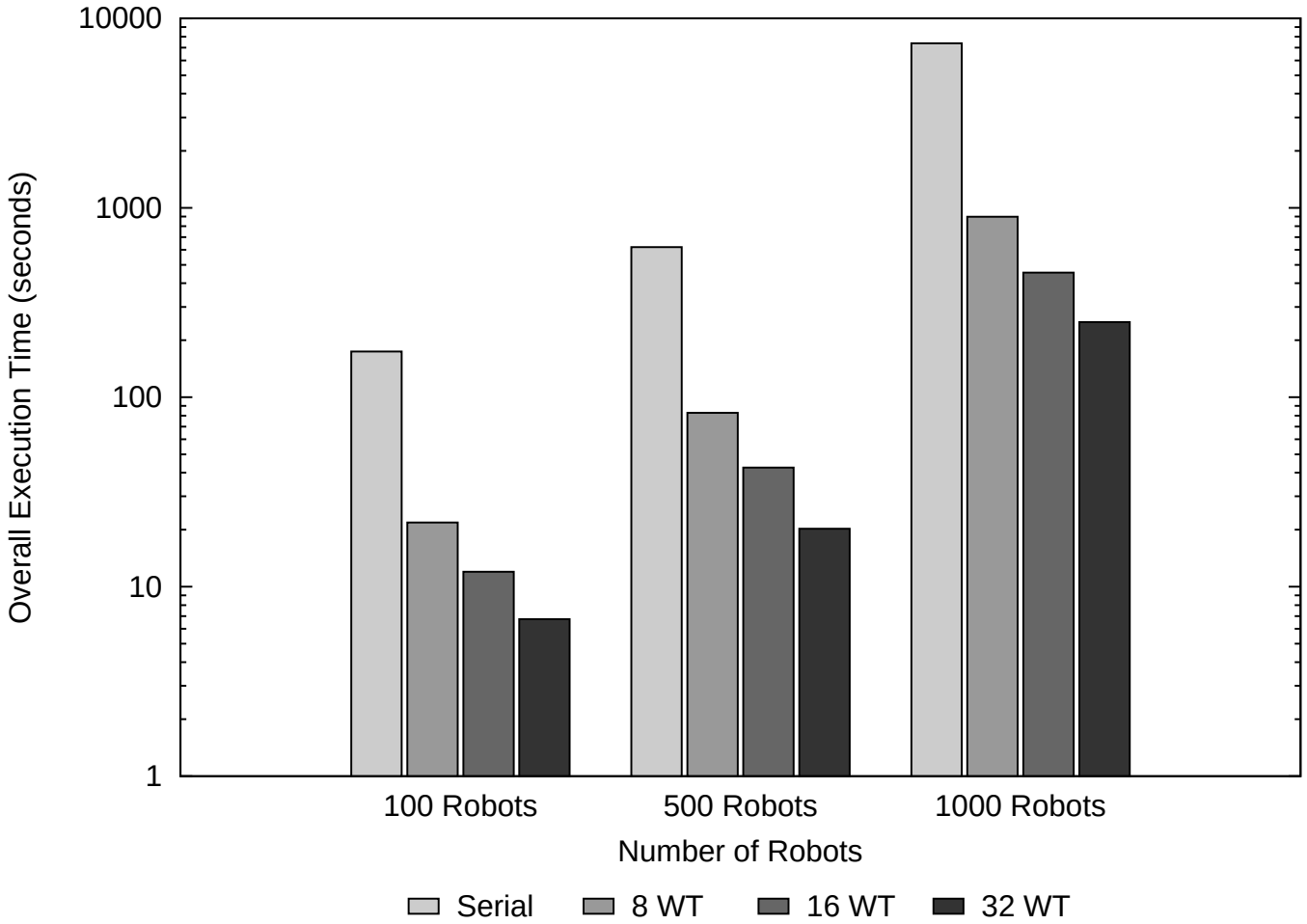
```
1 robot = (agent_state_type *)states[robot_index];
2 for(j = 0; j < num_cells; j++) {
3     if(robot->visit_map[j].visited) {
4         memcpy(&state->visit_map[j], &robot->visit_map[j], sizeof(
              map_t));
5     } else if (state->visit_map[j].visited) {
6         memcpy(&robot->visit_map[j], &state->visit_map[j], sizeof(
              map_t));
7     }
8 }
```

Performance Evaluation: Set up

- Hardware configuration:
  - HP ProLiant server equipped with 64GB of RAM
  - 4 8-cores CPU (32 cores total)

- Benchmark configuration:
  - 4096 cells
  - Variable number of robots: 100, 500, 1000
  - Serial simulation, vs Parallel Simulation using 8, 16, 32 cores

Performance Evaluation: Results



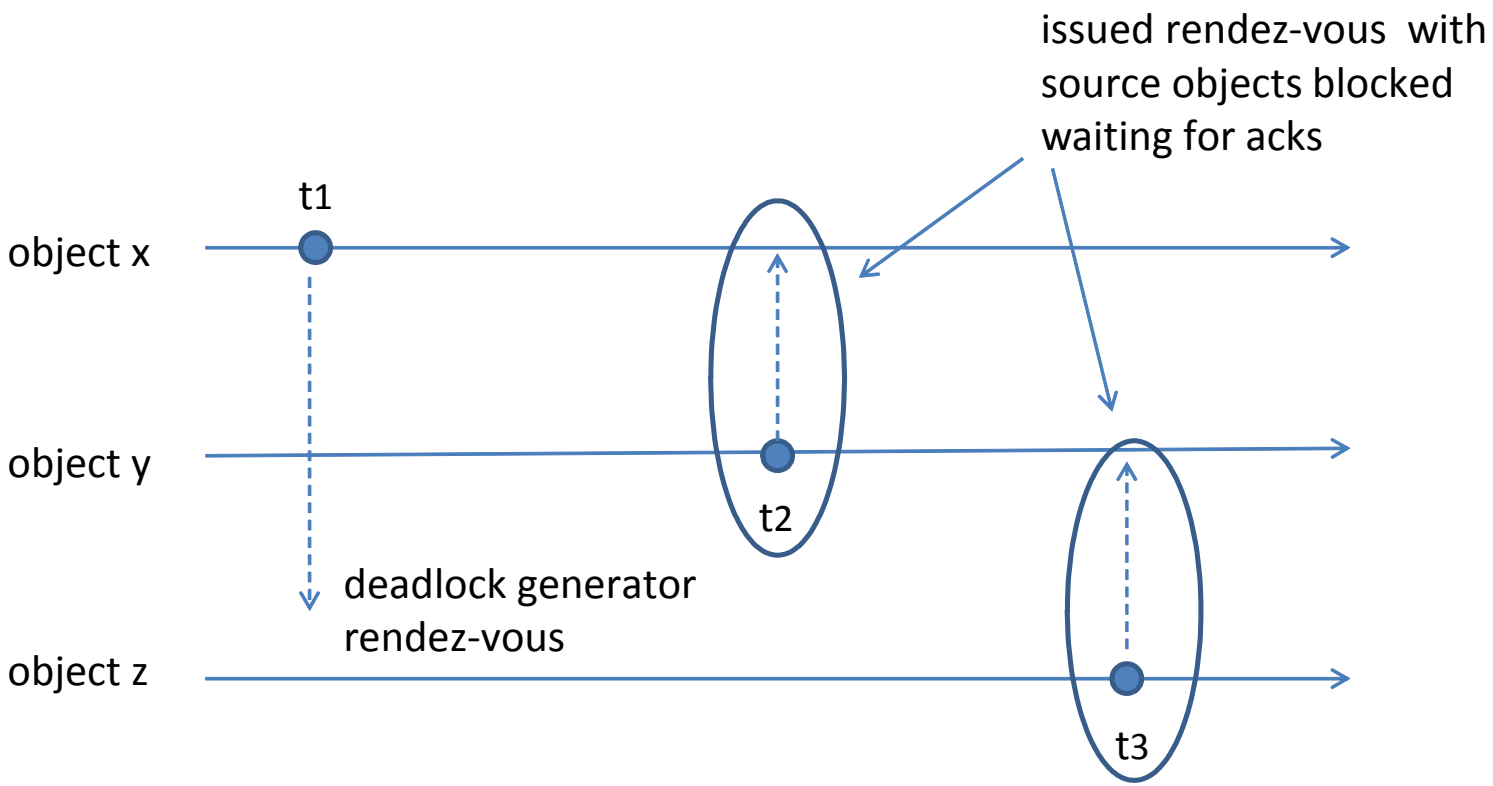
Thanks for your attention

pellegrini@dis.uniroma1.it
http://www.dis.uniroma1.it/∼pellegrini
http://www.dis.uniroma1.it/∼ROOT-Sim

Rollback

- Rollback of LP_x is managed via traditional annihilation scheme
- Rollback of LP_y must be explicitly notified
  - A restart event e_x^{rvr} is sent to LP_x
- All other events are not incorporated in the queue
  - They do not require special care for rollback operations
  - They are simply discarded if no rendez-vous ID is found

Progress: Deadlock



Progress: Domino Effect