(Un)Suspend(able)

Mark Burton

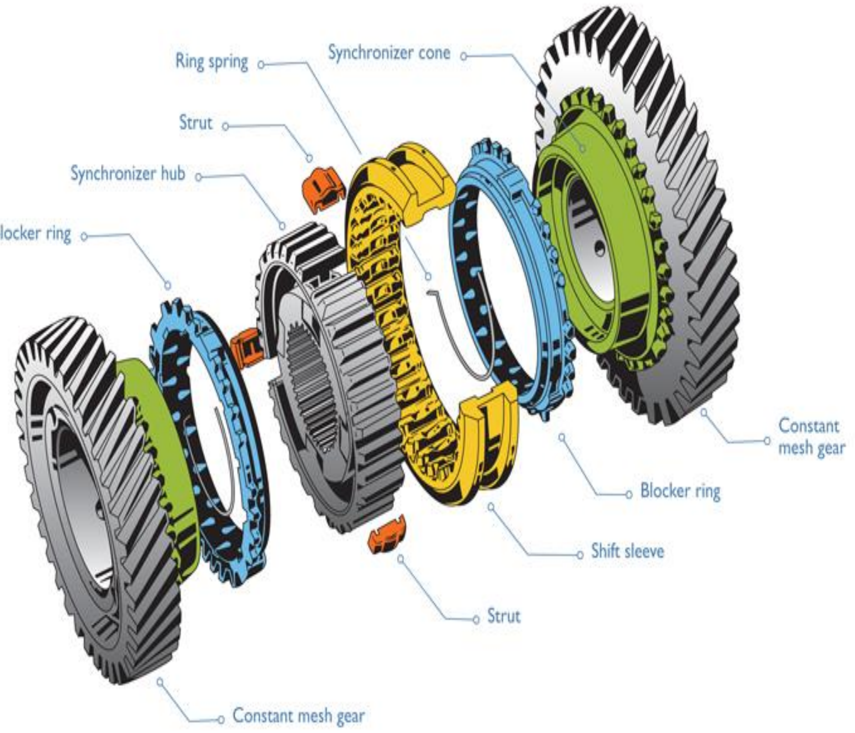
GreenSocs

EVOLUTION DAY
OCT 31, 2019 | MUNICH | GERMANY

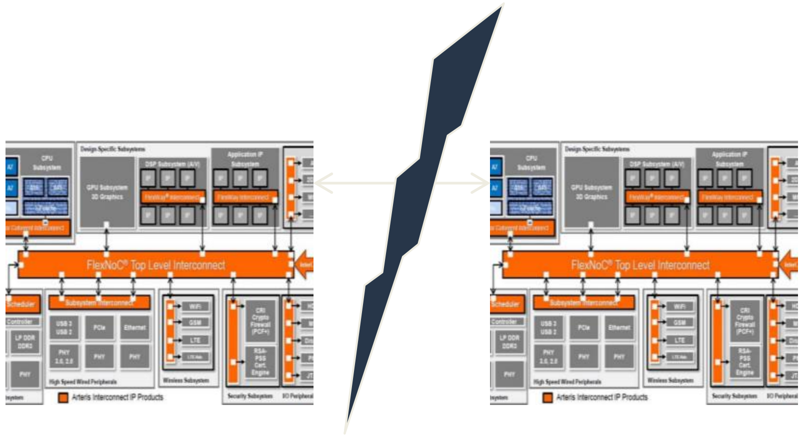
Presentation Copyright Permission

Synchronizing simulators and Save and Restore!

Syncromesh – it's all about connections between things moving at different speeds!



Motivation 1: Multiple Threads (and processes)

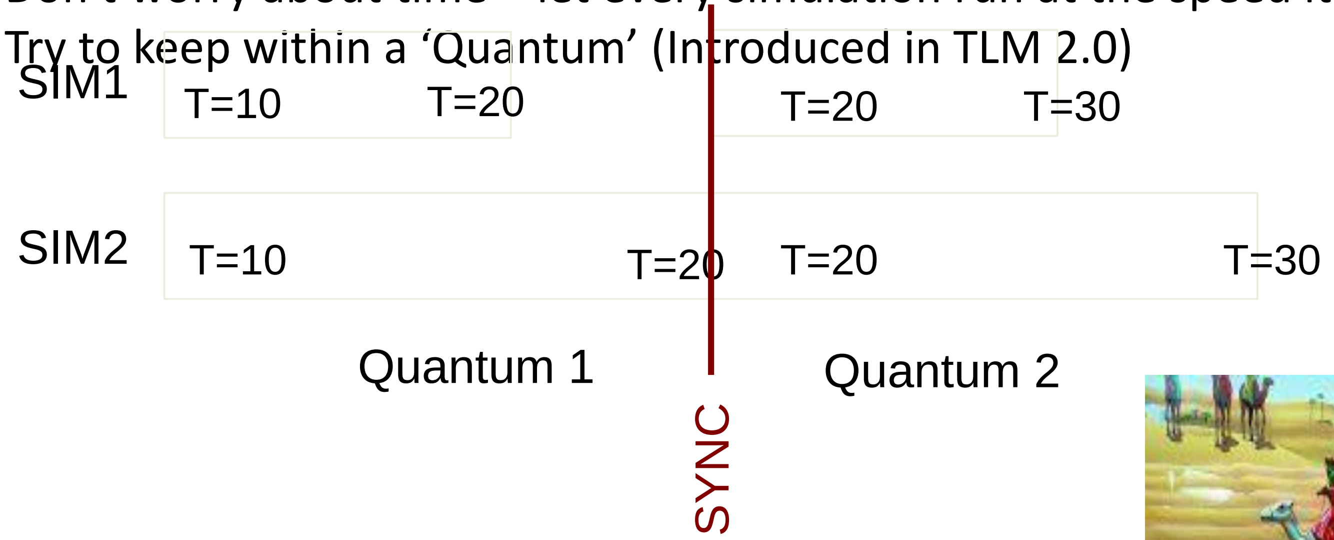

FlexNoC® Top Level Interconnect
Arteris Interconnect IP Products

We are talking about divide simulations based on architectural features (like CPU's or Ethernet)

(not on a SystemC thread/method level)

Time (Stops for nobody?)

- We have some choices:
  - Don’t worry about time – let every simulation run at the speed it wants!
  - Try to keep within a ‘Quantum’ (Introduced in TLM 2.0)



  - SIM1 runs ‘faster’ than SIM2.
  - At least once per quantum, the simulations are synchro
  - SIM1 needs to ‘wait’ for SIM2.

SUSPEND!

- We need a mechanism to **SUSPEND** a simulation

  while it waits for the other simulations catch up (and send an event to continue).

- **NB ‘suspend’ exists (!) for individual threads.**
- We need to suspend all threads, so that SystemC has nothing more to do.

Motivation 2: Save/Restore when it's safe

Stop when all threads when you are not in a unsuspendable state.

NB b_transport has no way to re-start – no way to rebuild the stack.

SystemC 2.3.1…

SOME BACKGROUND…

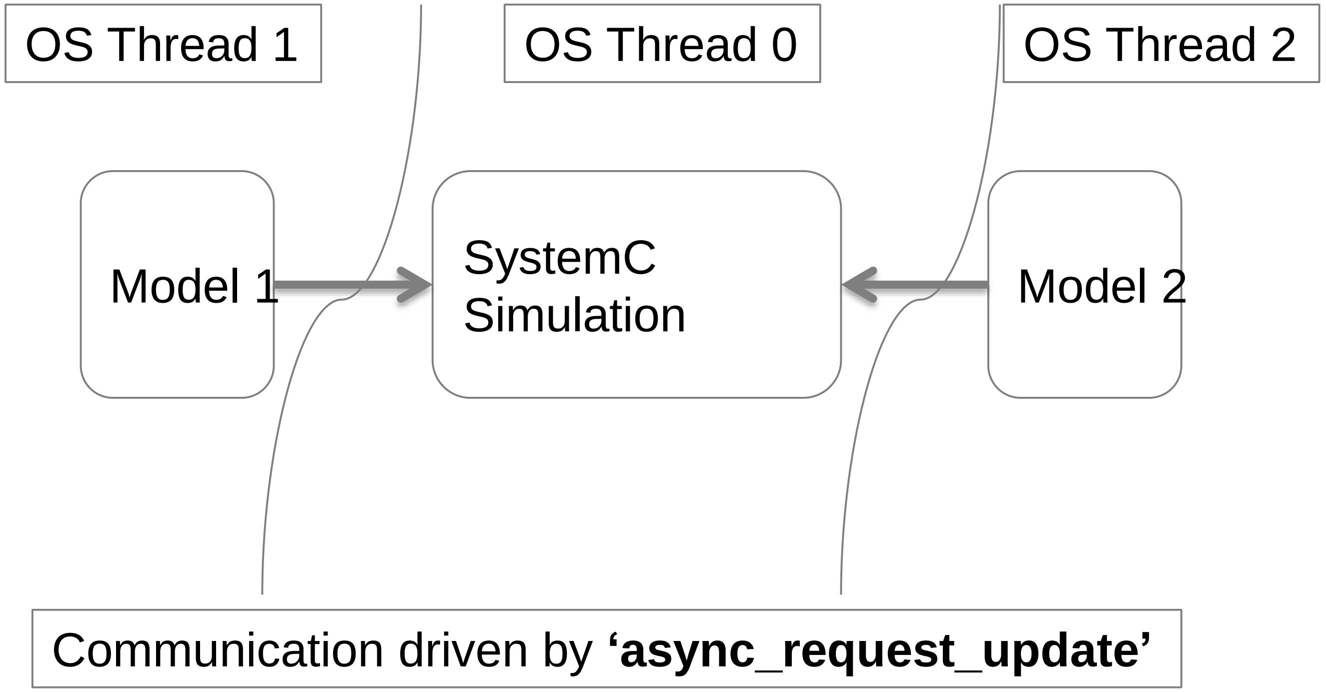
async_request_update

- **async_request_update** – allows an external ‘event’ to be inserted into a running SystemC kernel in a thread safe way.

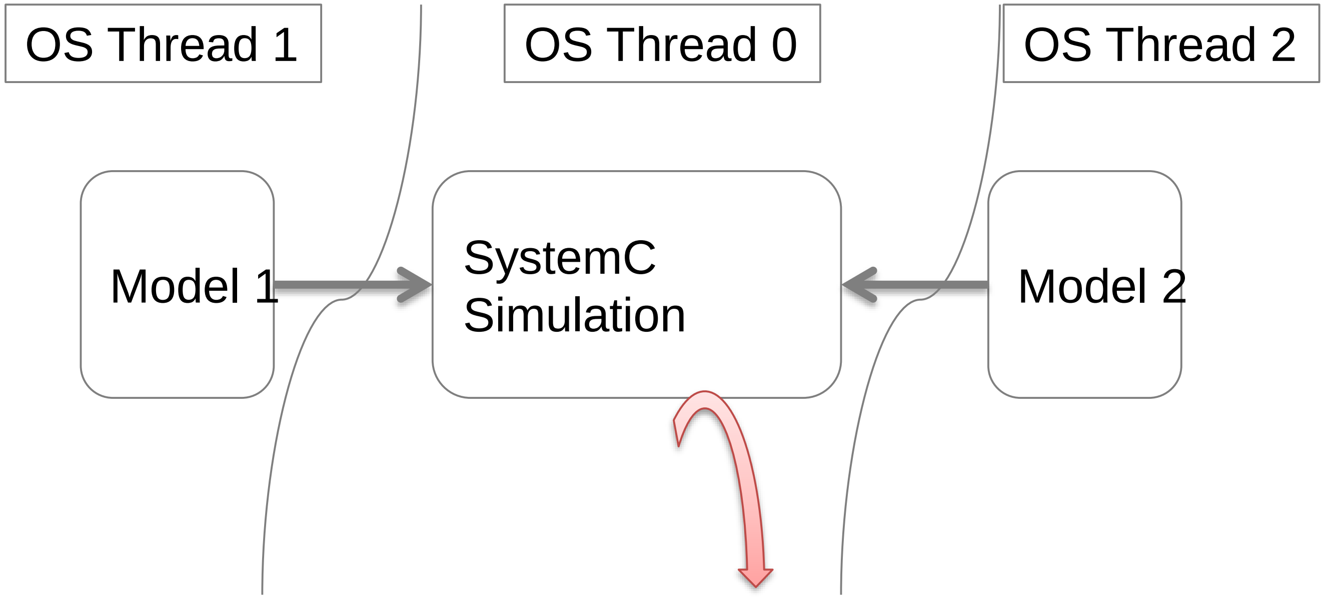
- Basis of any (all) communication between two simulations

- Problem : If a simulation runs out of events, then …
  we better not stop!
  (And if we have suspended all the threads, that can happen a lot!)
- And we need a single common semaphore

The situation



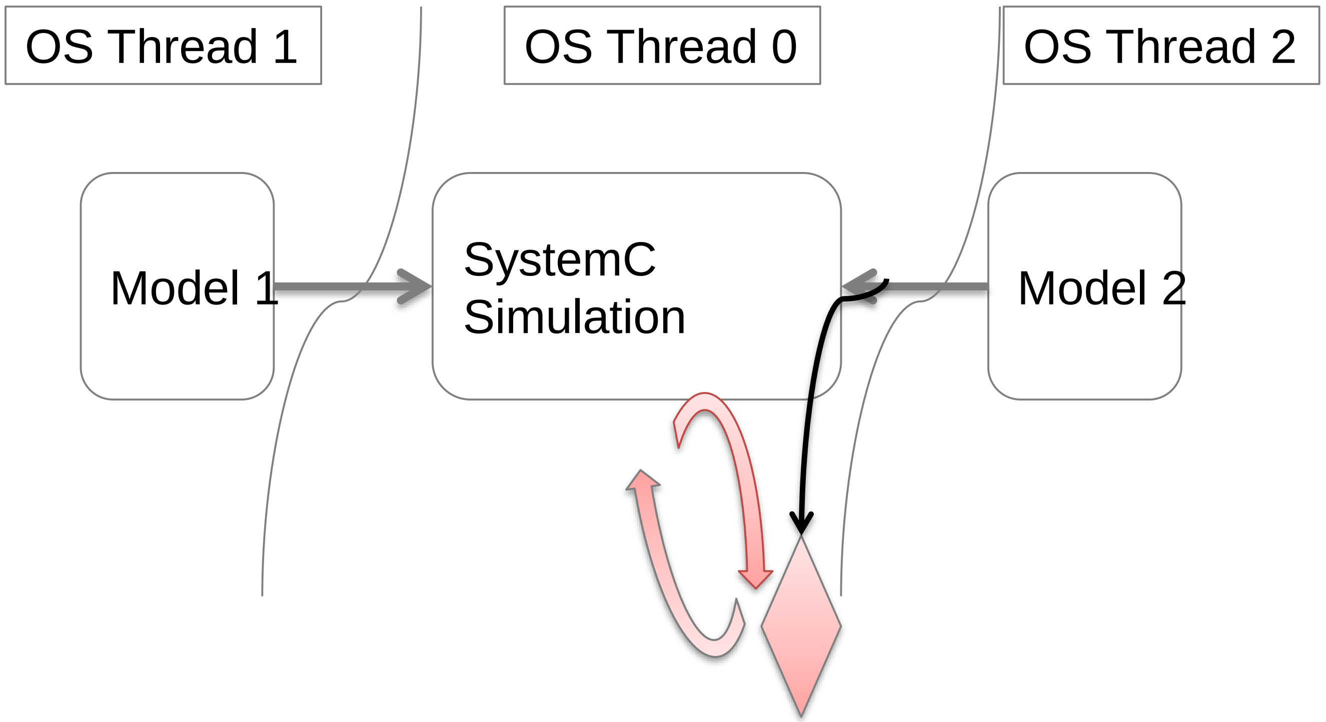
The problem



SystemC runs out of events…
(even though model 1 and 2 are still active)

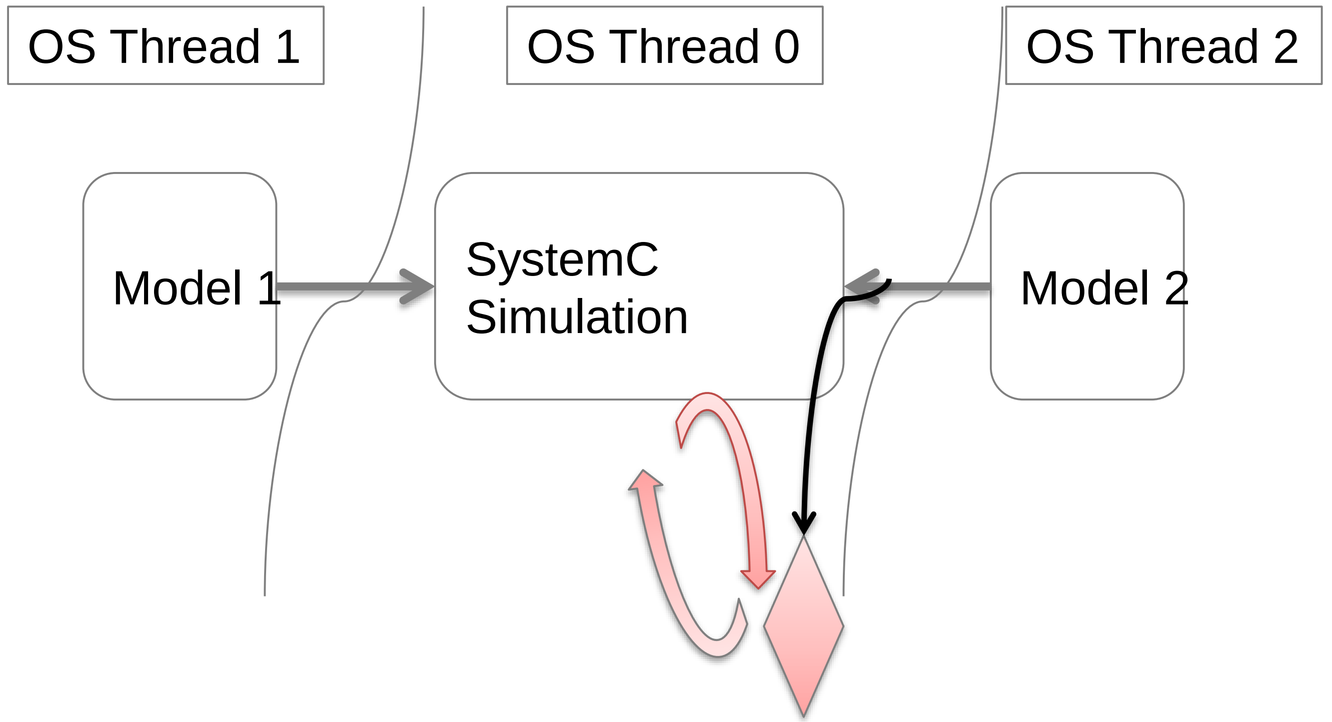
Simulation ***dies***.

Solution (#1)



Catch SystemC ‘just before’ it finishes in a semaphore,
Release the semaphore on an event

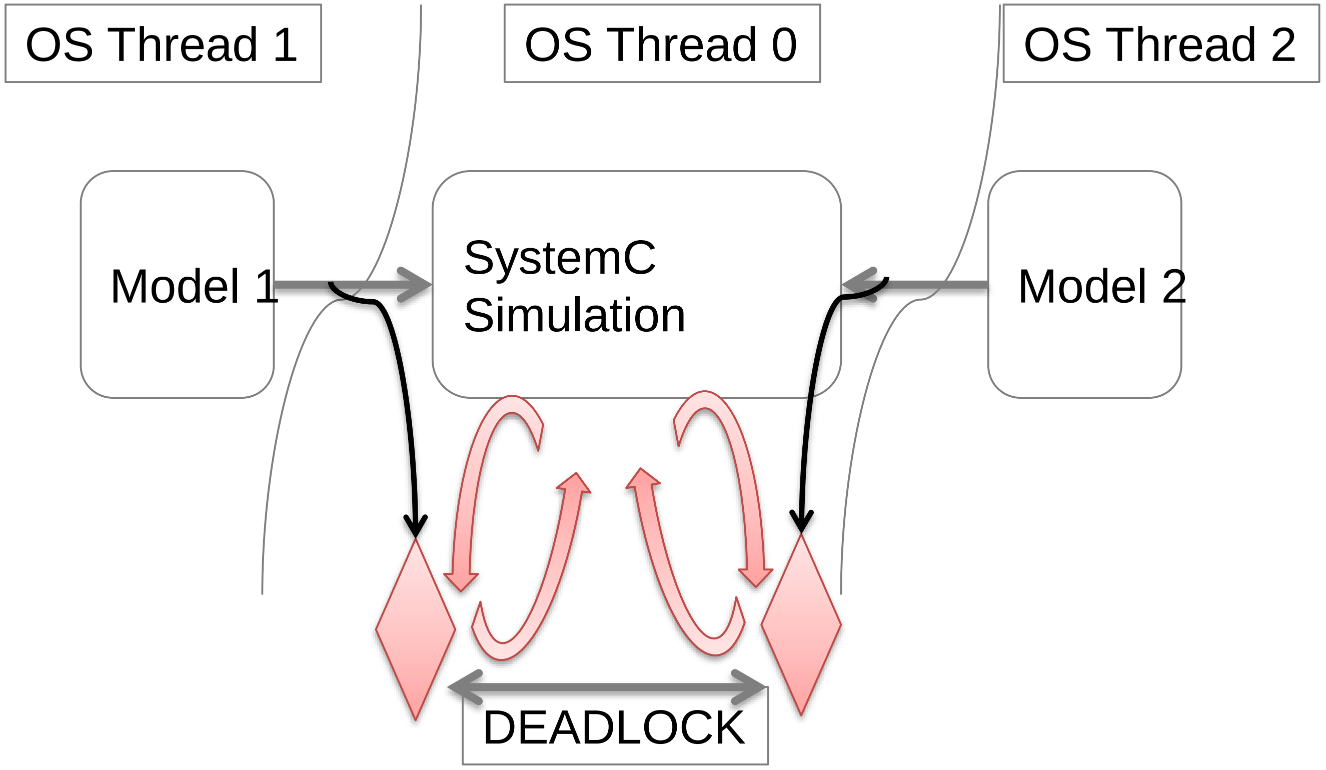
Solution (#1)



In SystemC 2.3.1, to catch SystemC just before it finishes,
You can post an event into the next delta
(using **sc_time_to_pending_activity**)

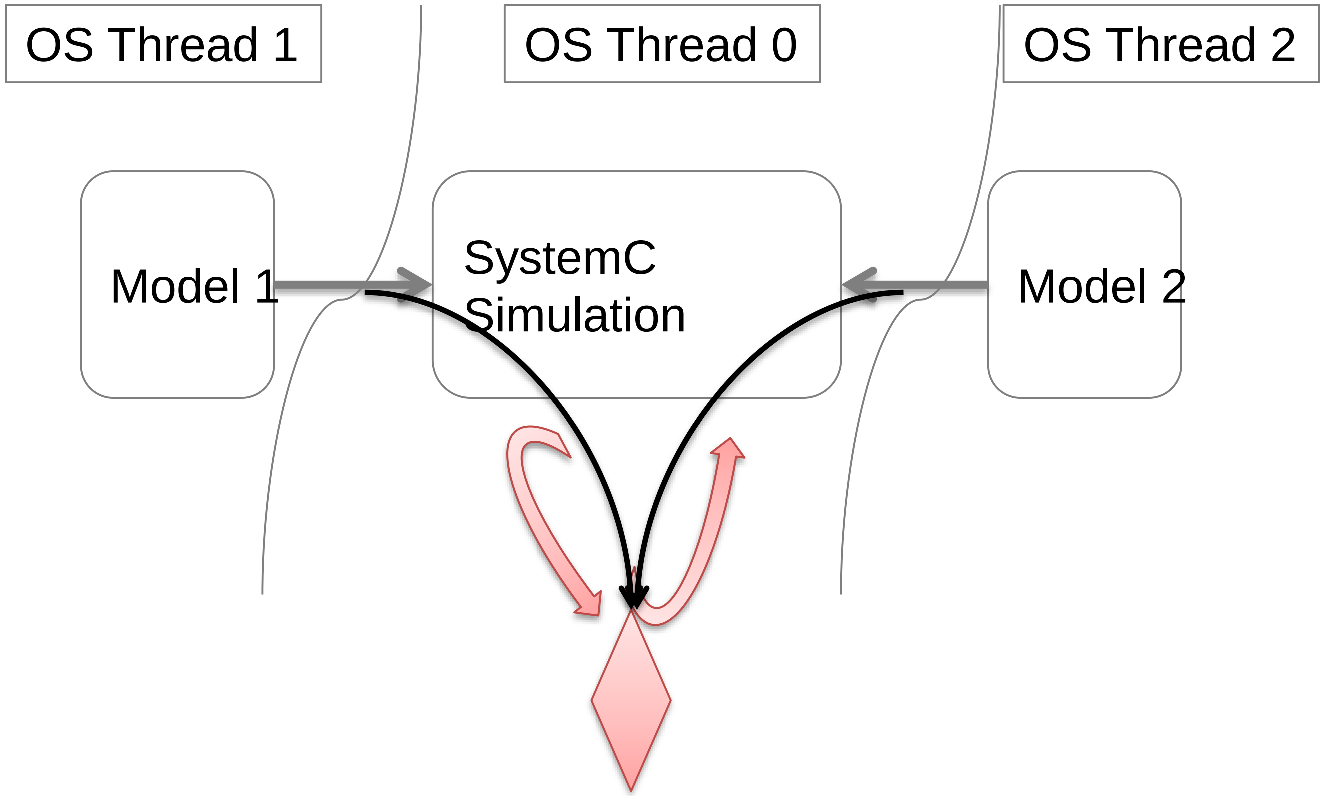
It adds an extra notification (at least) to each delta. ☹

Solution is not compose-able



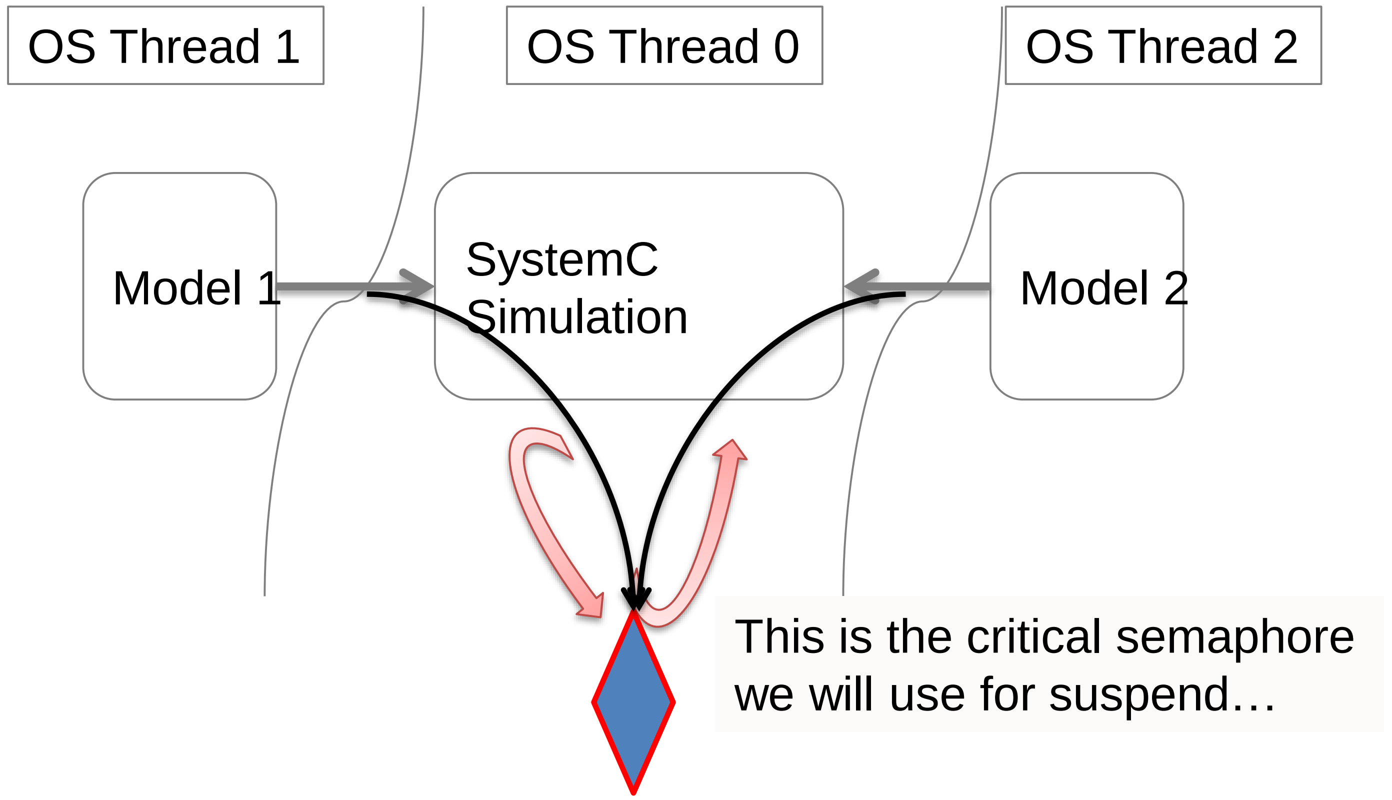
For more than one model (from different sources) the solution will ***fail*** ☹

Compose-able solution for **2.3.2**



Single shared semaphore.
Triggered from **async_request_update**

Compose-able solution for **2.3.2**



This is the critical semaphore we will use for suspend…

Single shared semaphore.
Triggered from **async_request_update**

Details (Available in SystemC 2.3.2+)

- By default the semaphore is not used, SystemC exits normally.
- **bool async_attach_suspending** // proposed for IEEE 1666-202x
  - Prim_channels can elect to attach to an external source of events (and therefore request the presence of the semaphore)
- **bool async_detach_suspending**
  - A prim_channel can elect to detach from an external source of events (and therefore remove the request for the presence of the semaphore). If no prim_channels are attached to external events, the semaphore plays no role in simulation.
- The semaphore is only checked (and potentially waited for) if no further events are available
  - starvation, which is checked inside the simulation kernel anyway (using **next_time** returns 0), so there is no additional cost here.
- The semaphore is released when **async_request_update** is called.

Building on central semaphore

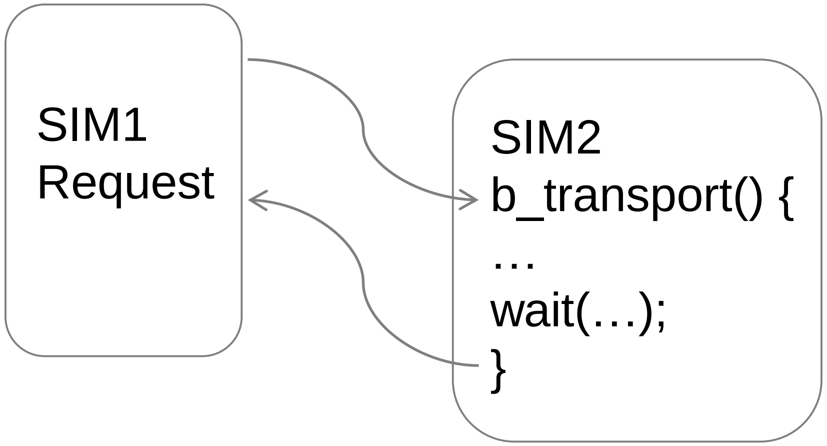
SUSPEND

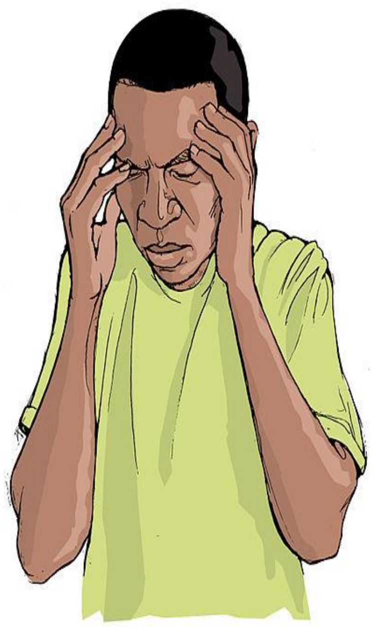
Suspend

- Add a kernel function
- suspend_all()
  - Request to suspend all threads (and pending events)
  - All threads become un-schedulable
  - (The simulation will run out of schedulable tasks instantly, and fall into the 'semaphore').
- Unsuspend()
  - Request to re-instate all threads (and pending events).
  - *You can call unsuspend from a async_update method as you will not be in the semaphore at that point.*

BUT… b_transport and wait() !

- Same old **b_transport** wait pain !





- If SIM2 is 'ahead' of SIM1, and is currently 'suspended', then we will be BLOCKED DEADLOCK

unsuspendable

- Need a mechanism to mark **‘b_transport’s** that come from an external simulation as “unsuspendable”.

HOLD THAT THOUGHT

…

SAVE and RESTORE

- The CCI WG has a proposal to support save/restore.
- An API that will be added to SystemC modules
- Ensure that models can be re-entered from a ‘restore’ call.

- BUT – what happens if a model is in a **‘b_transport’**, that has called a **‘wait’** ?
- **‘b_transport’** is not restartable on the current transaction.

- We need an ‘un-savable at this point’ mechanism !!!

Unsuspendable/Suspendable

- A thread may mark itself as ‘unsuspendable’.
- This only effects the ‘global’ suspend_all mechanism.
- For save/restore, all b_transports that are non-re-entrant will have to be non-suspendable.
- For thread sync, all b_transports being processed on behalf of an external simulation should be marked as non-suspendable.

Save/restore + suspendable

- To do a save:
  - Do a **‘suspend_all’**, followed by a **‘save’**

- To do a restore:
  - Do a **‘restore’**, followed by a **‘unsuspend_all’**

- We use **suspend_all**/**unsuspend_all** to make sure that its “safe” to **save**/**restore**. (Models are not inside b_transport’s that have called **‘wait’**).

- Of course it’s possible to make a simulation that will never suspend
- Some work will be required to make sure models don’t always call a blocking **‘b_transport’**.

new API

- The existing suspend/resume API does not include a ‘nesting’ mechanism, and it’s not possible to add it in user-space.
- An additional API is required, the question is WHAT?
- Current proposal:

```
void sc_suspend_all(sc_simcontext* csc=sc_get_curr_simcontext)
void sc_unsuspend_all(sc_simcontext* csc=sc_get_curr_simcontext)
void sc_unsuspendable()
void sc_suspendable()
```

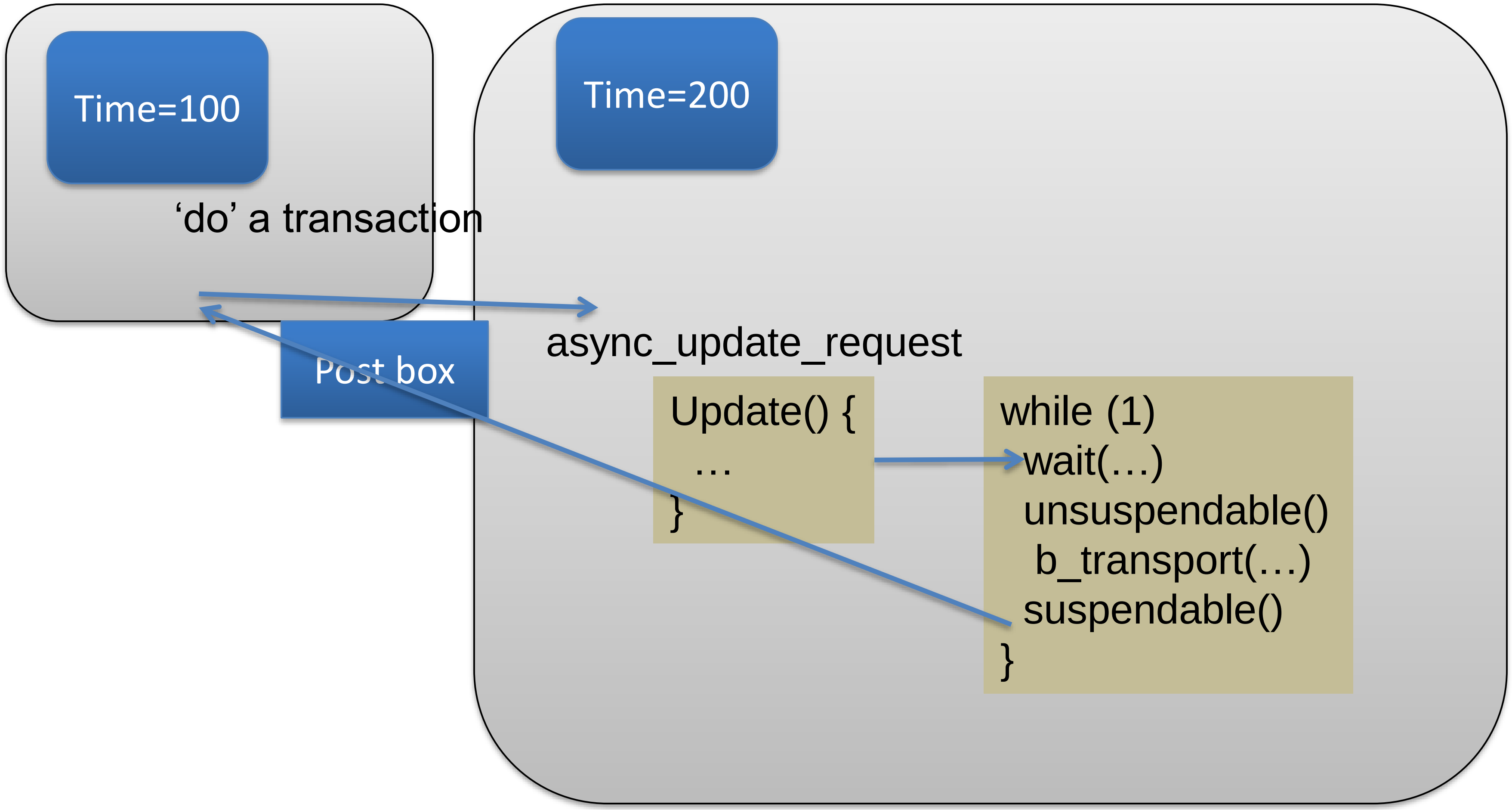
suspend_all/unsuspend_all :

- Requests the kernel to ‘**atomically suspend**’ all processes (using the same semantics as the thread suspend() call). This is atomic in that the kernel will only suspend all the processes together, such that they can be suspended and **unsuspended without any side effects**.
  - Calling suspend_all(), and subsiquently calling unsuspend_all() will have no effect on the suspended status of an individual thread.
- A process may call suspend_all() followed by unsuspend_all, the calls should be ‘paired’, (multiple calls to either suspend_all() or unsuspend_all() will be ignored).
- Outside of the context of a process, it is the programmers responsibility to ensure that the calls are paired.
- As a consequence, multiple calls to suspend_all() may be made (within separate processes, or from within sc_main). So long as there have been more calls to suspend_all() than to unsuspend_all(), the kernel will suspend all processes.

unsusbendable()/suspendable()

- This pair of functions provides an ‘opt-out’ for specific process to the suspend_all. The consequence is that if there is a process that has opted out, the kernel will not be able to suspend_all (as it would no longer be atomic).
- These functions can only be called from within a process.
- A process should only call suspendable/unsuspendable in pairs (multiple calls to either will be ignored).

Expected sequence



```
Update() {
  …
}
```

```
while (1)
  wait(…)
  unsuspendable()
   b_transport(…)
  suspendable()
}
```

Additional proposals

- dont_terminate A way to wrap a thread with a

  while(1) {wait(trigger) … }

  - Such a non terminating thread is (probably?) unsuspendable.
- NS_THREAD = a SC thread that is sensitive to an event (like a cthread), and who's method is marked as unsuspendable (it will always complete).
- We also need is_suspending/ed()
- Some sort of "suspended()" simulation stat callback

Status

- Initial patch exists
- But – we need help !

- This is going to be a big change to the way SystemC can be used ! It's important for large simulations to take advantage of parallelism.
- Please be in touch with Mark Burton @ GreenSocs.com who is leading this effort.