Barcelona Supercomputing Center
Centro Nacional de Supercomputación

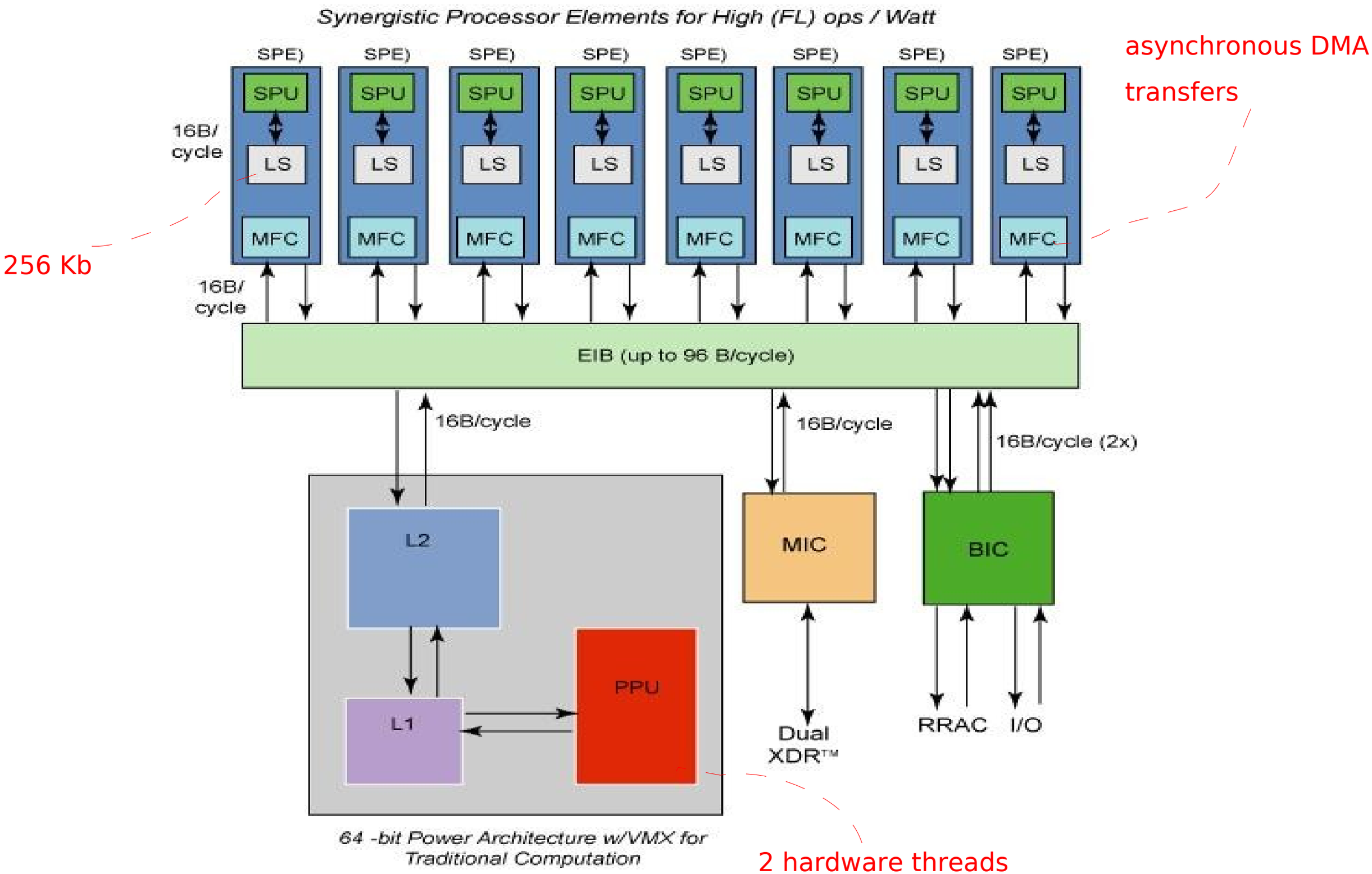
JIT renaming and lazy write-back on the Cell/B.E.

Pieter Bellens, Josep M. Perez,
Rosa M. Badia, Jesus Labarta
Barcelona Supercomputing Center (BSC-CNS)
pieter.bellens@bsc.es

Overview

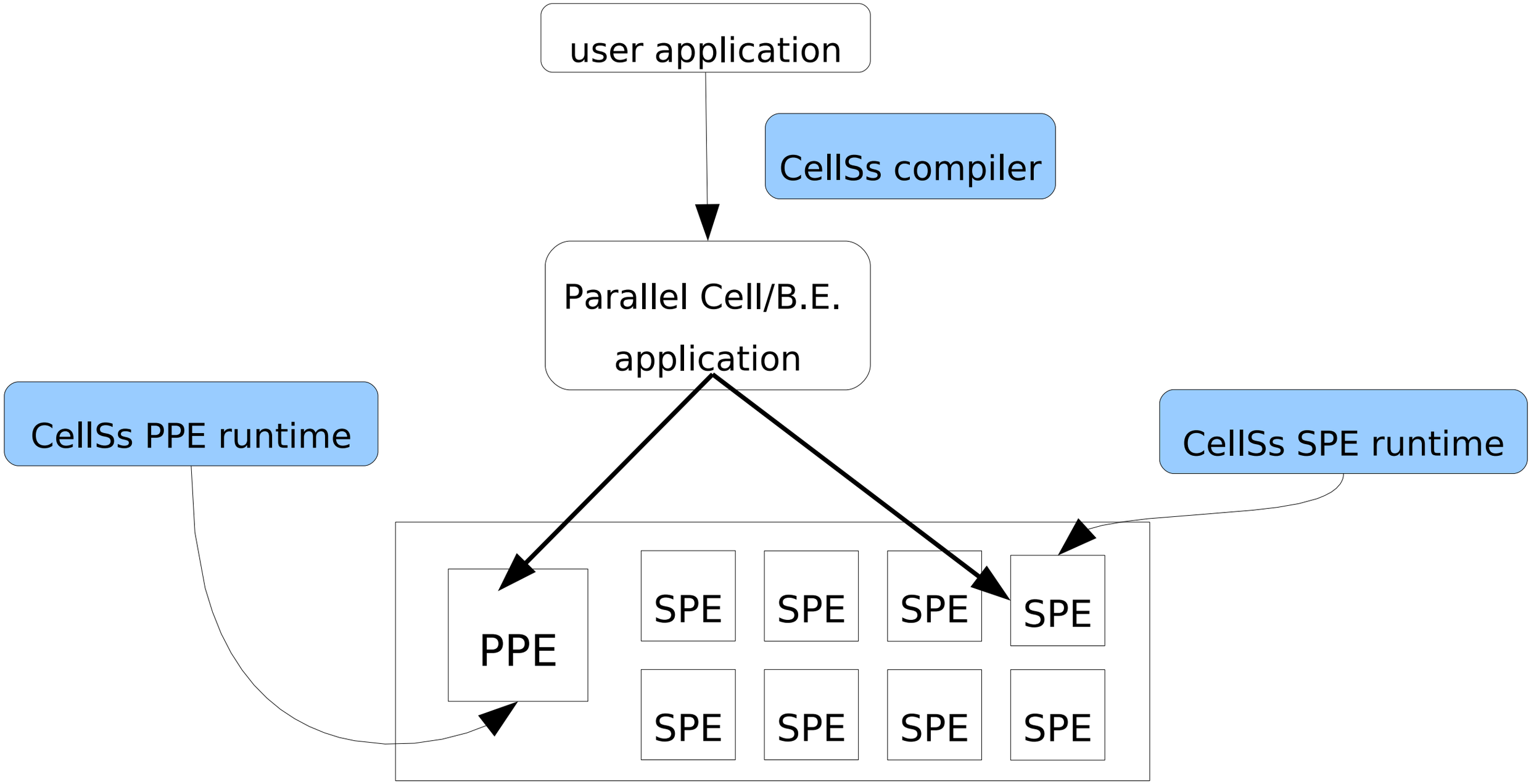
- Cell Broadband Engine (Cell/B.E.)
- Cell Superscalar (CellSs)
- Bypassing
  - Motivation
  - Implementation
  - Results
- Lazy write-back
- Just-In-Time renaming
- Current status and ongoing work

Cell Broadband Engine



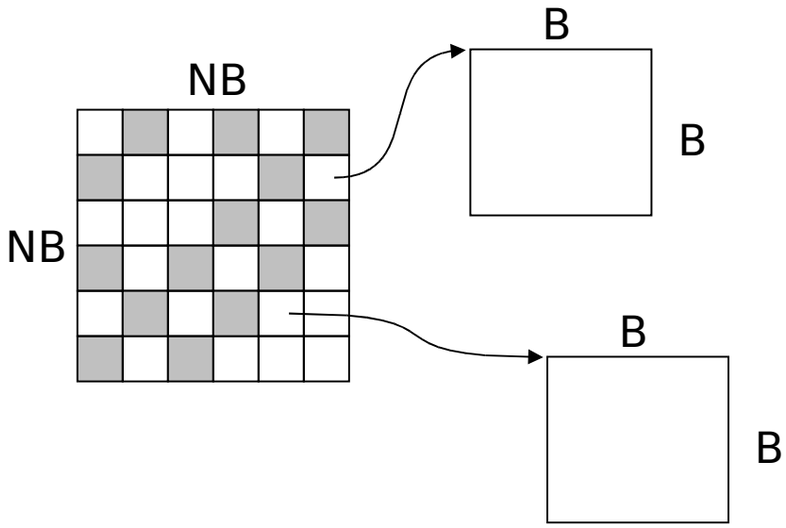
CellSs

Runtime environment that automatically parallelizes sequential user applications for the Cell/B.E.



CellSs: sample code (sparse LU)

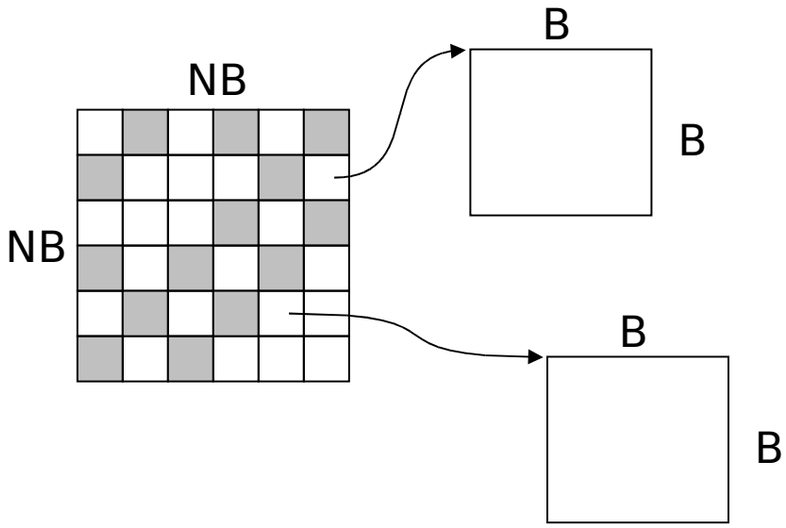
```
int main(int argc, char **argv) {
int ii, jj, kk;
…

 for (kk=0; kk<NB; kk++) {
    lu0(A[kk][kk]);
      for (jj=kk+1; jj<NB; jj++)
        if (A[kk][jj] != NULL)
           fwd(A[kk][kk], A[kk][jj]);
      for (ii=kk+1; ii<NB; ii++)
        if (A[ii][kk] != NULL) {
          bdiv (A[kk][kk], A[ii][kk]);
            for (jj=kk+1; jj<NB; jj++)
              if (A[kk][jj] != NULL) {
                if (A[ii][jj]==NULL)
                  A[ii][jj]=allocate_clean_block();
                  bmod(A[ii][kk], A[kk][jj], A[ii][jj]);
               }
        }
    }
}
```




```
void lu0(float *diag);

void bdiv(float *diag, float *row);


void bmod(float *row, float *col, float *inner);

void fwd(float *diag, float *col);
```

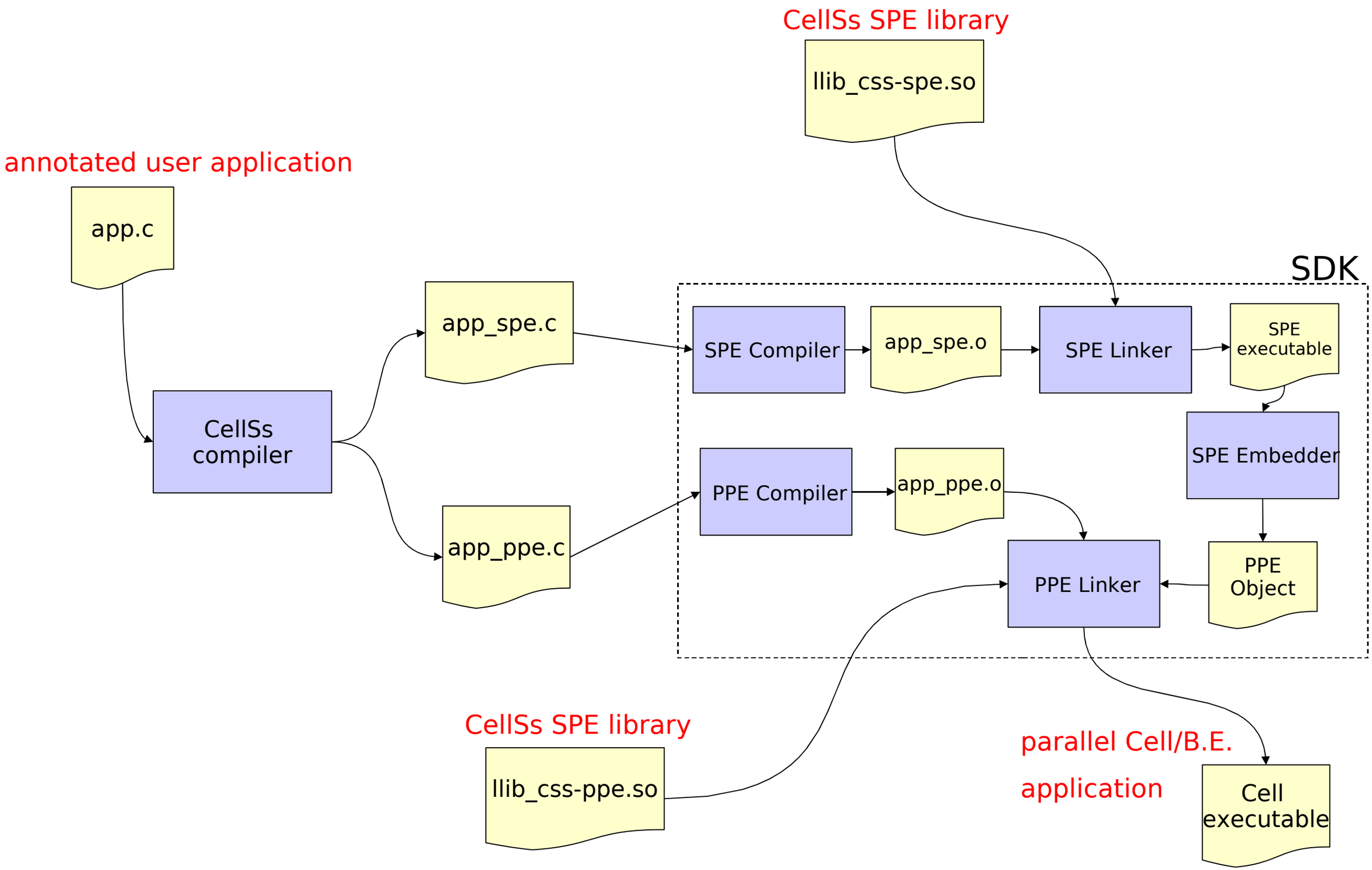
CellSs: sample code (sparse LU)

```
int main(int argc, char **argv) {
int ii, jj, kk;
…

 for (kk=0; kk<NB; kk++) {
    lu0(A[kk][kk]);
      for (jj=kk+1; jj<NB; jj++)
        if (A[kk][jj] != NULL)
           fwd(A[kk][kk], A[kk][jj]);
      for (ii=kk+1; ii<NB; ii++)
        if (A[ii][kk] != NULL) {
          bdiv (A[kk][kk], A[ii][kk]);
            for (jj=kk+1; jj<NB; jj++)
              if (A[kk][jj] != NULL) {
                if (A[ii][jj]==NULL)
                  A[ii][jj]=allocate_clean_block();
                  bmod(A[ii][kk], A[kk][jj], A[ii][jj]);
               }
        }
    }
}
```

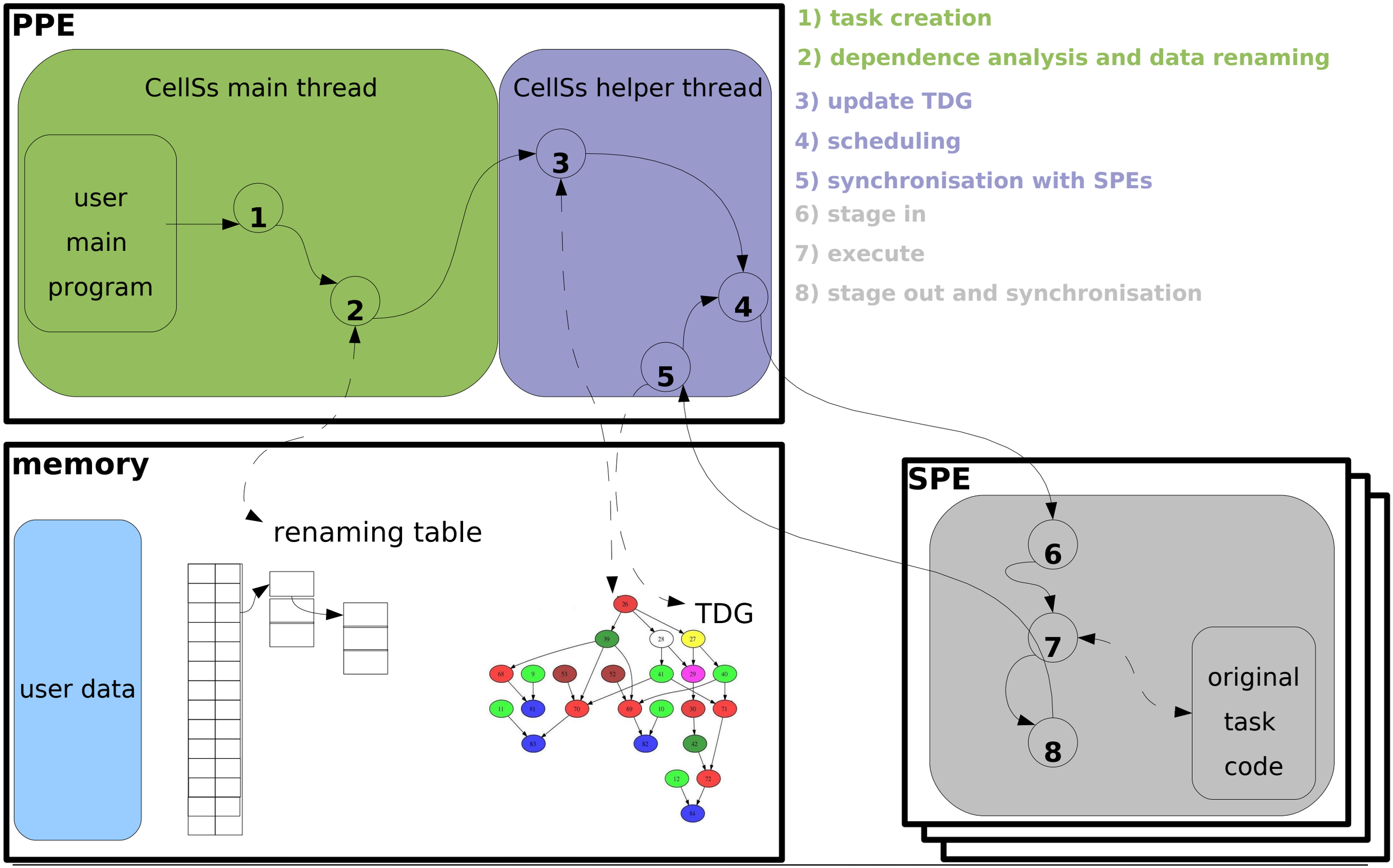



```
#pragma css task inout(diag[B][B])
void lu0(float *diag);
#pragma css task input(diag[B][B]) inout(row[B][B])
void bdiv(float *diag, float *row);
#pragma css task input(row[B][B],col[B][B]) inout(inner[B][B])
void bmod(float *row, float *col, float *inner);
#pragma css task input(diag[B][B]) inout(col[B][B])
void fwd(float *diag, float *col);
```

CellSs: compiler

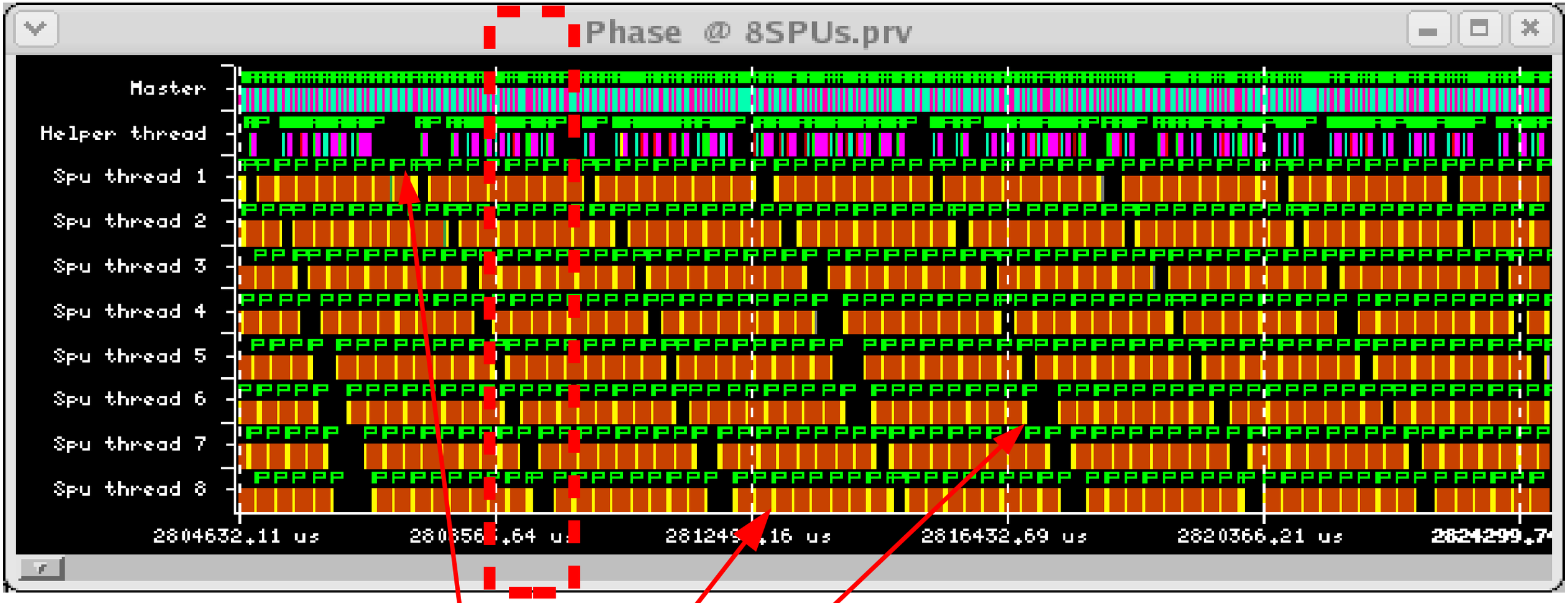


CellSs: runtime libraries



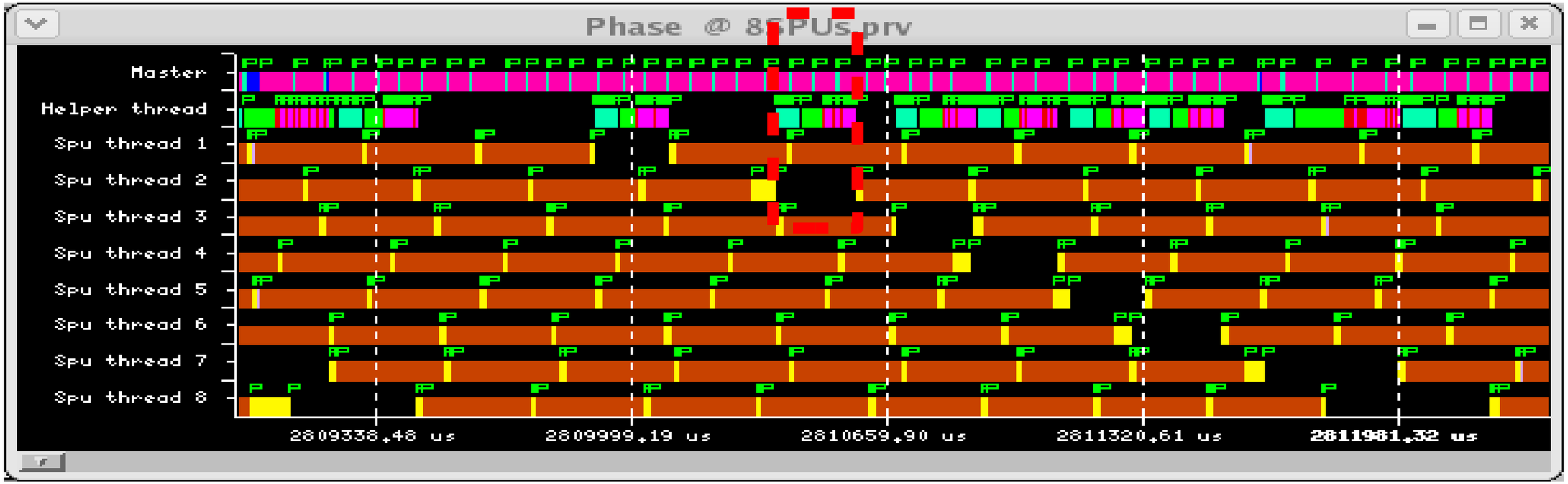
1) task creation
2) dependence analysis and data renaming
3) update TDG
4) scheduling
5) synchronisation with SPEs
6) stage in
7) execute
8) stage out and synchronisation

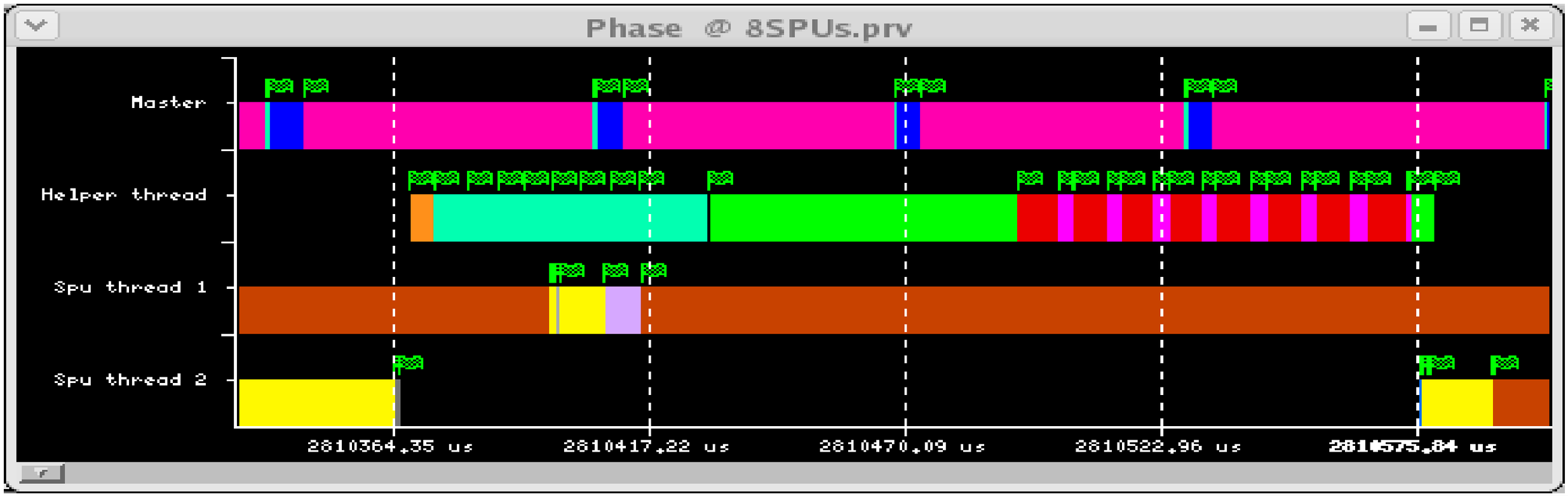
CellSs: runtime behaviour (matrix multiply)



- Visualization of the runtime phases in function of time using Paraver
- Each phase is assigned a different colour
  - SPE task execution
  - SPE DMA wait
  - Thread idling

CellSs: runtime behaviour (matrix multiply)





Bypassing: motivation

A new architecture, but the song remains the same:

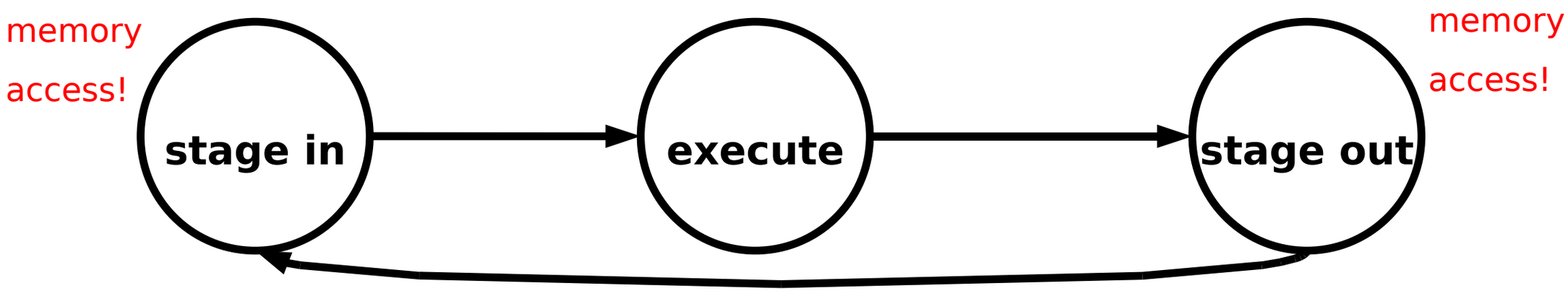
Improve the performance

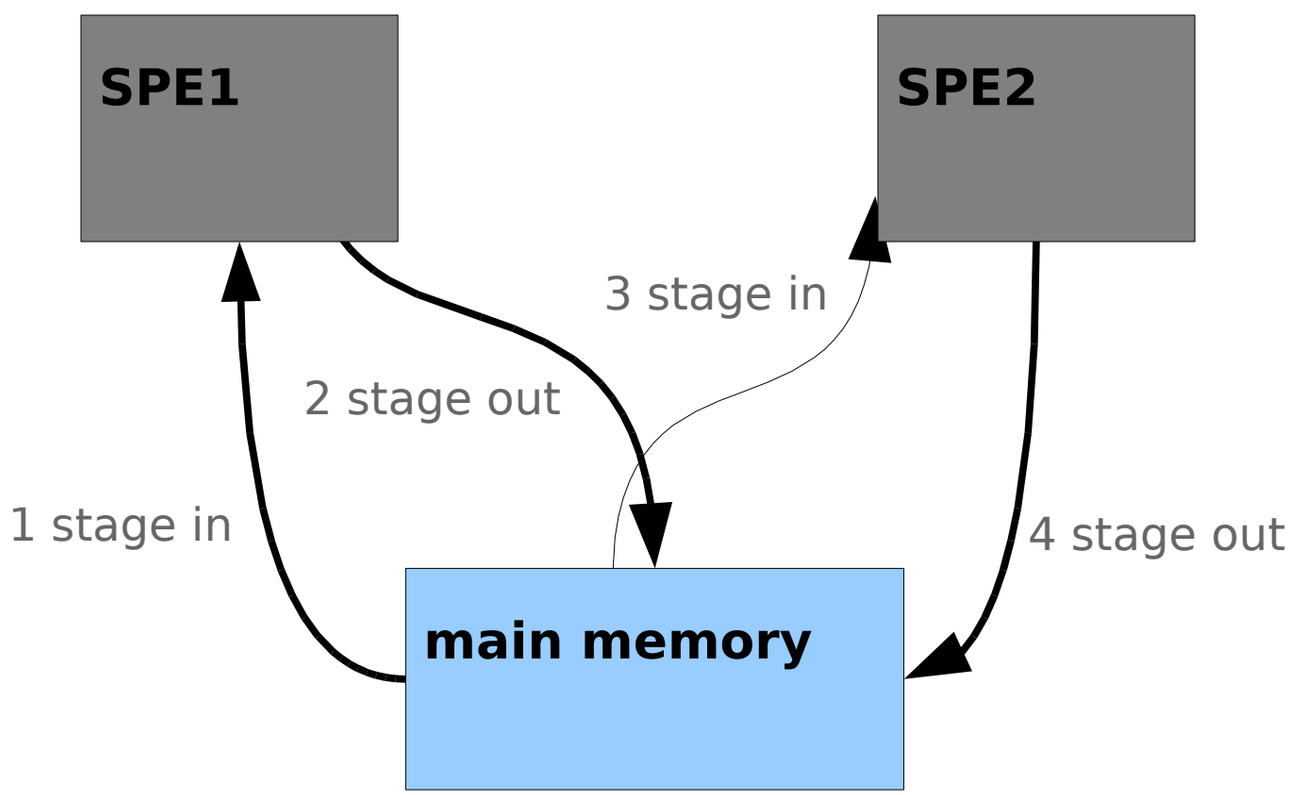
Let's take a closer look at code executing on the Cell/B.E.:

- General computation pattern
  - PPE generates work for SPEs
  - SPEs repeatedly fetch work and perform computation
    - Traditional approach vs. bypassing approach
- Cell/B.E. Interconnect
  - Element Interconnect Bus (EIB)

Bypassing: motivation: general computation pattern

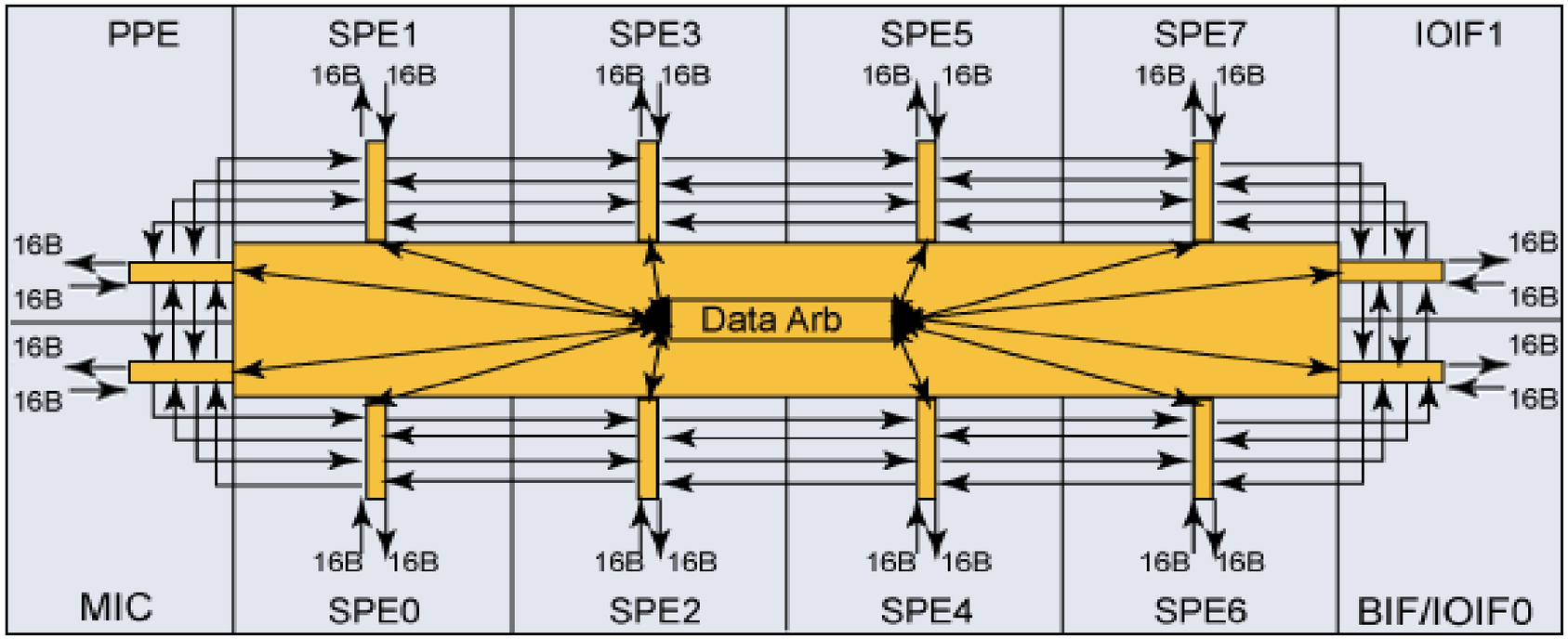
traditional:





Bypassing: motivation: Cell/B.E. interconnect

Element Interconnect Bus (EIB):

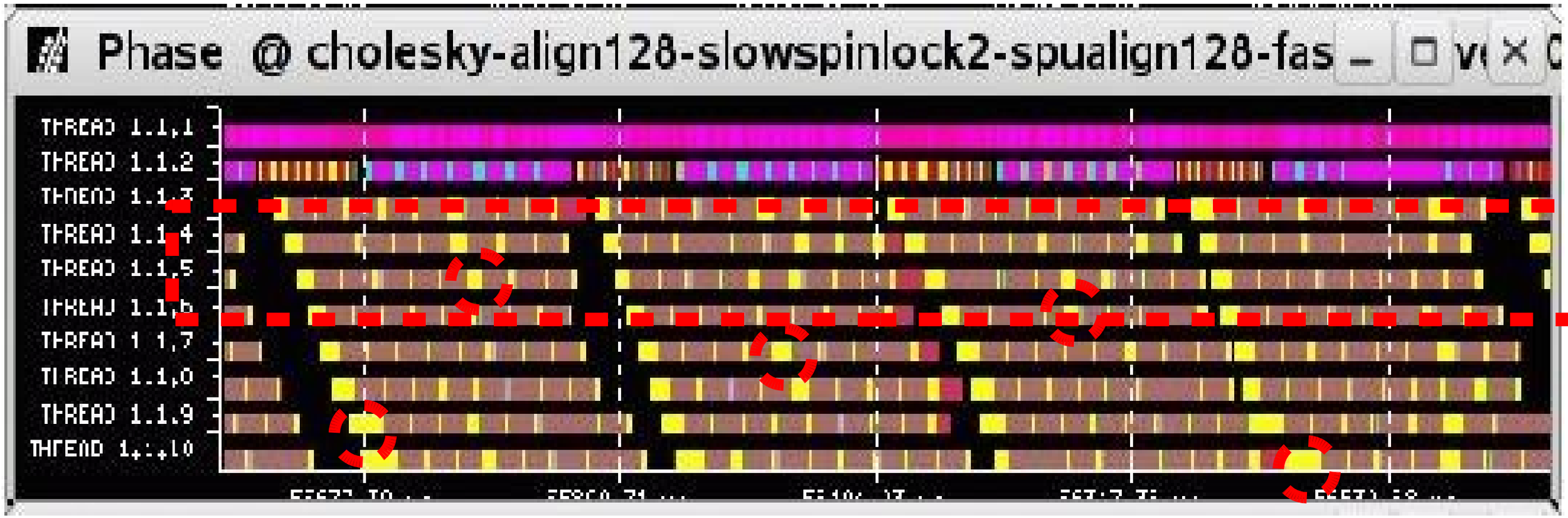


"Another class of bottlenecks is contention. For instance, if four SPEs are trying to move data to or from the MIC at the same time, their aggregate bandwidth of 102.4GB/sec completely swamps the MIC's bandwidth of 25.6GB/sec. Similarly, while the SPEs are trying to interact with the MIC, the PPE may have degraded access to main memory. When a unit is overwhelmed, it might need to retry commands, which in turn slows traffic down even further."

David Krolak, "Unleashing the Cell Broadband Engine Processor: the Element Interconnect Bus"

Bypassing: motivation

How does contention and blocking influence the execution?

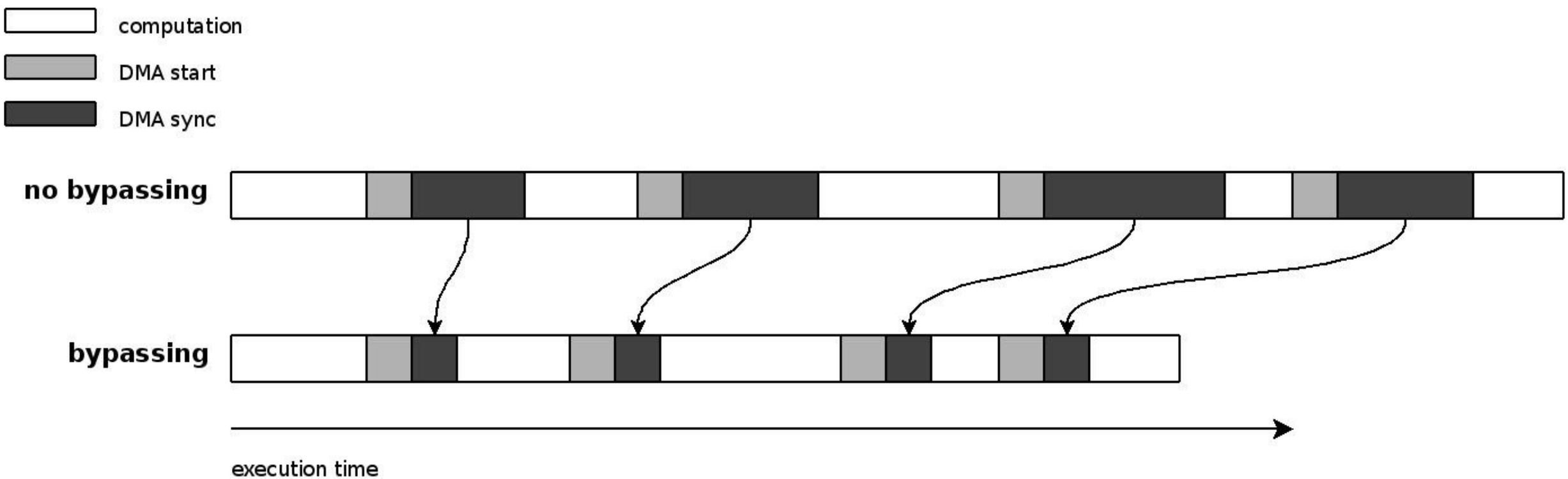


- Countermeasures:
  - software cache in the LS of an SPE
  - double buffering
  - ???

Bypassing: motivation

Transfer objects between the LS of SPEs without going through main memory

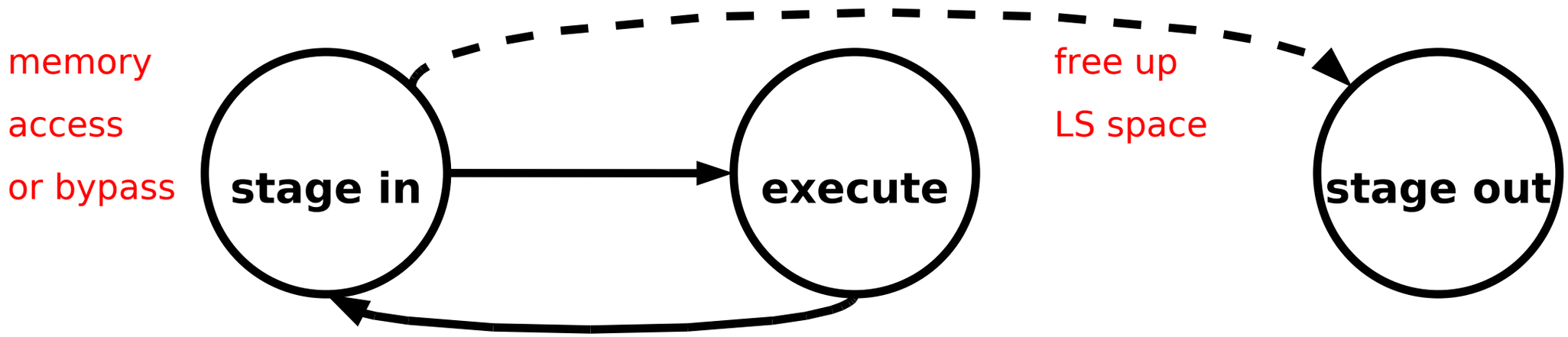
- General idea:

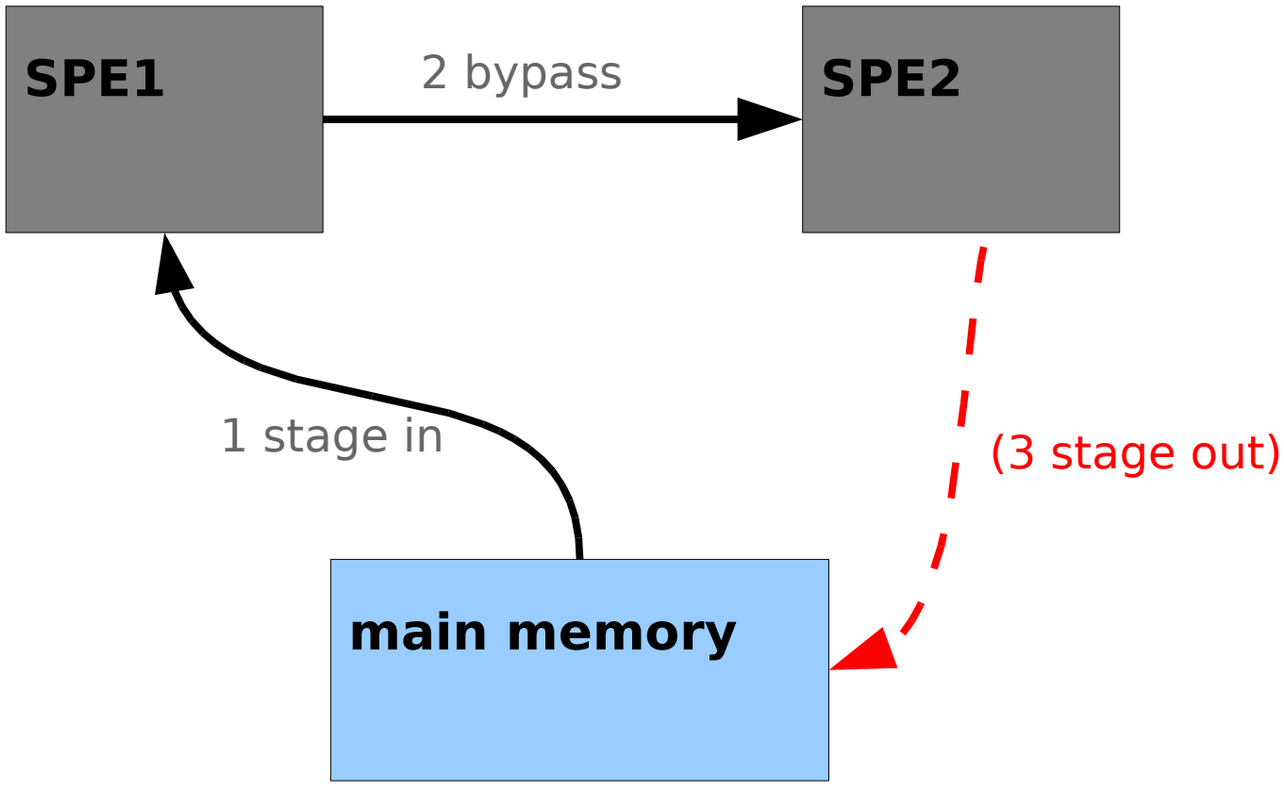


- Effect on PPE threads?

Bypassing: motivation

bypassing:



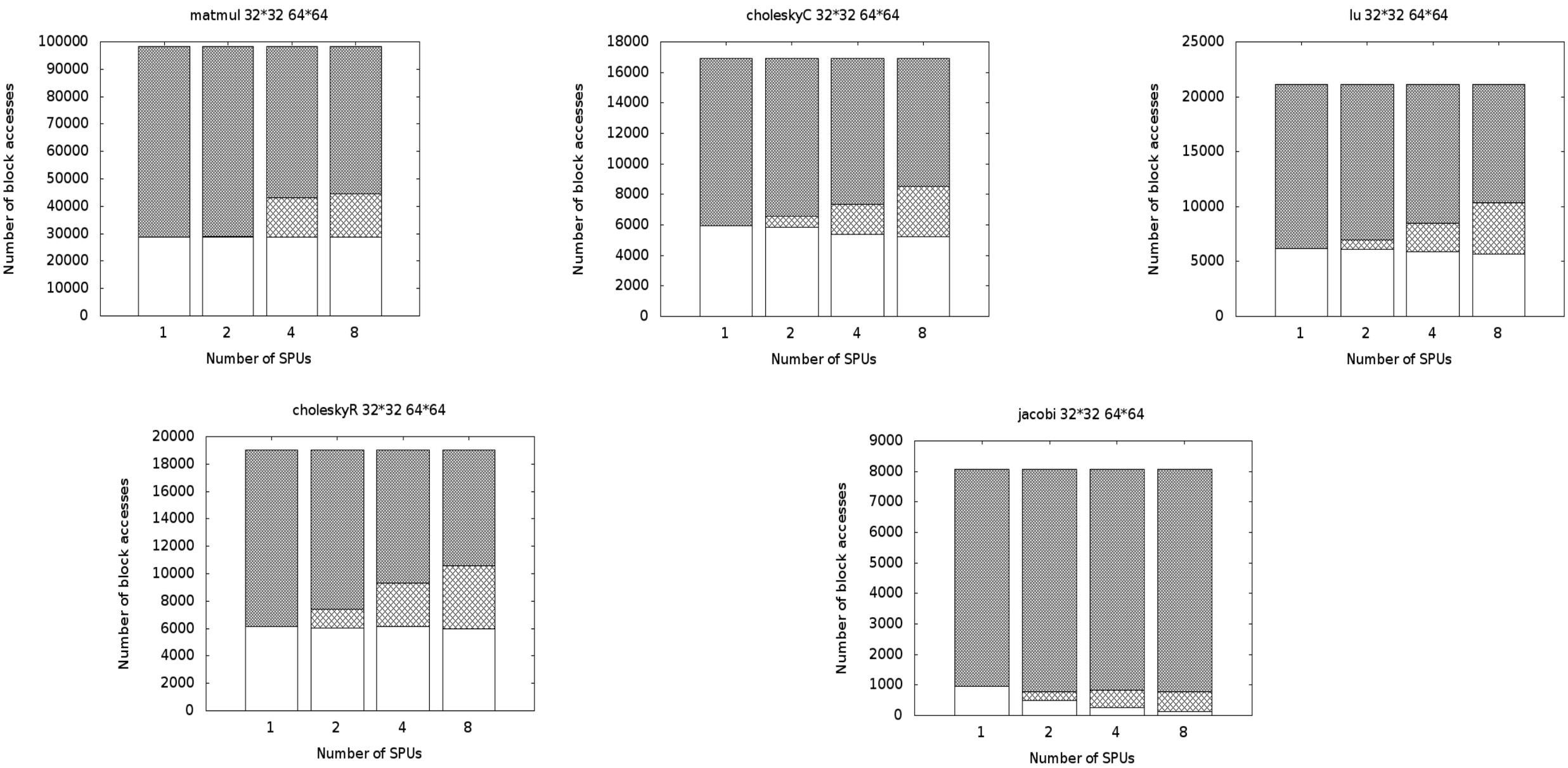


Bypassing: implementation

- General solution
  - SPE runtime autonomously decides to go to main memory or to bypass from another SPE
  - No need to tailor the bypassing mechanism to a specific application
- Implemented using the SPE's Atomic Cache Unit (ACU)
  - Location of software objects in the system is updated using the ACU
  - Distributed solution
  - Makes good use of hardware features

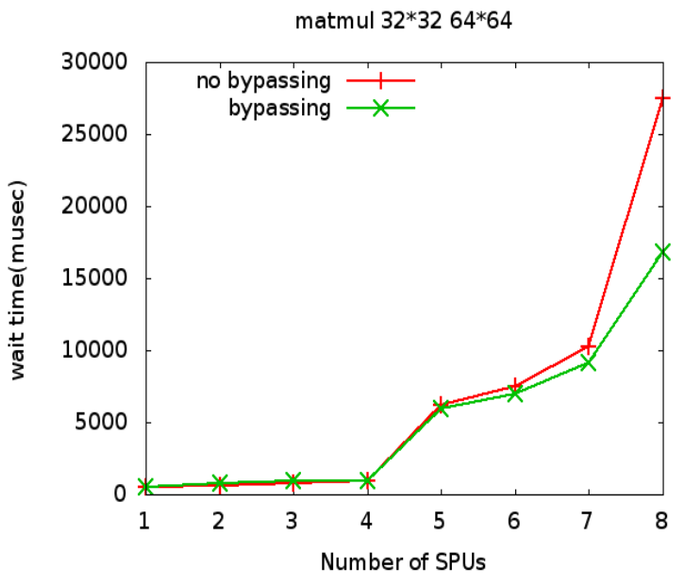
Bypassing: results: opportunities for bypassing

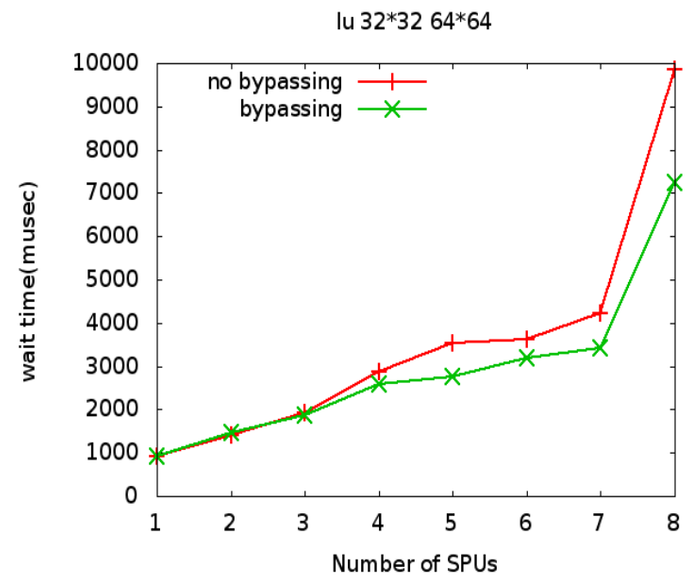
Are there opportunities to bypass data from one SPE to another?

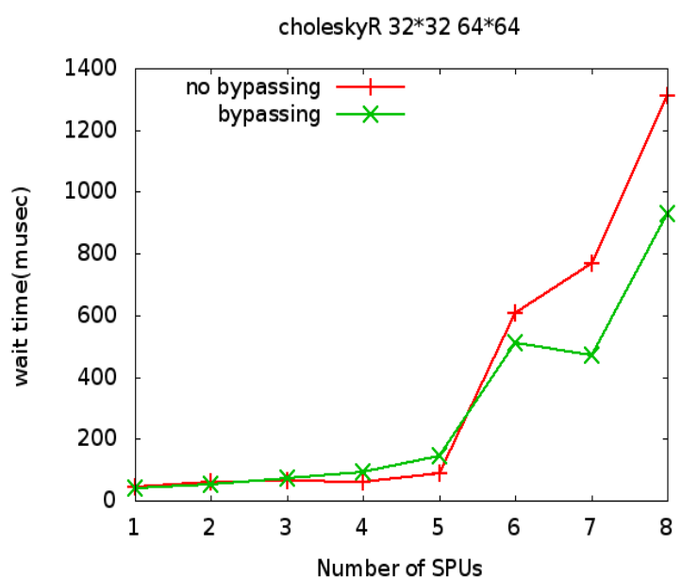


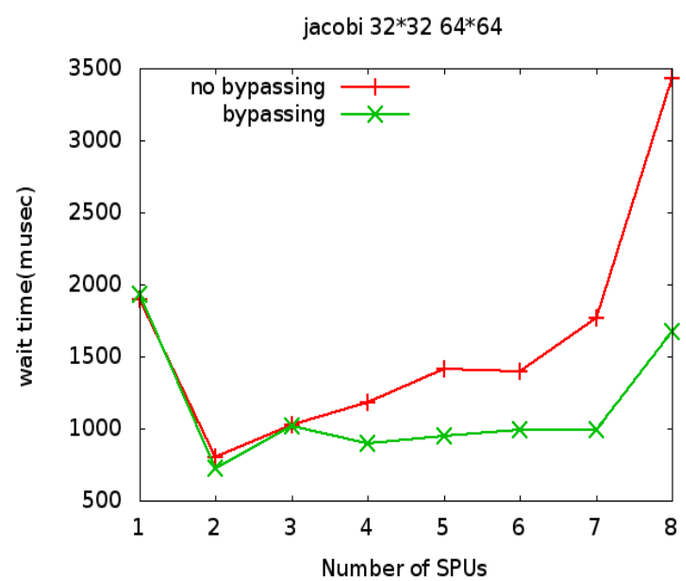
Bypassing: results: reduction in wait time

Does the wait time effectively decrease when bypassing?



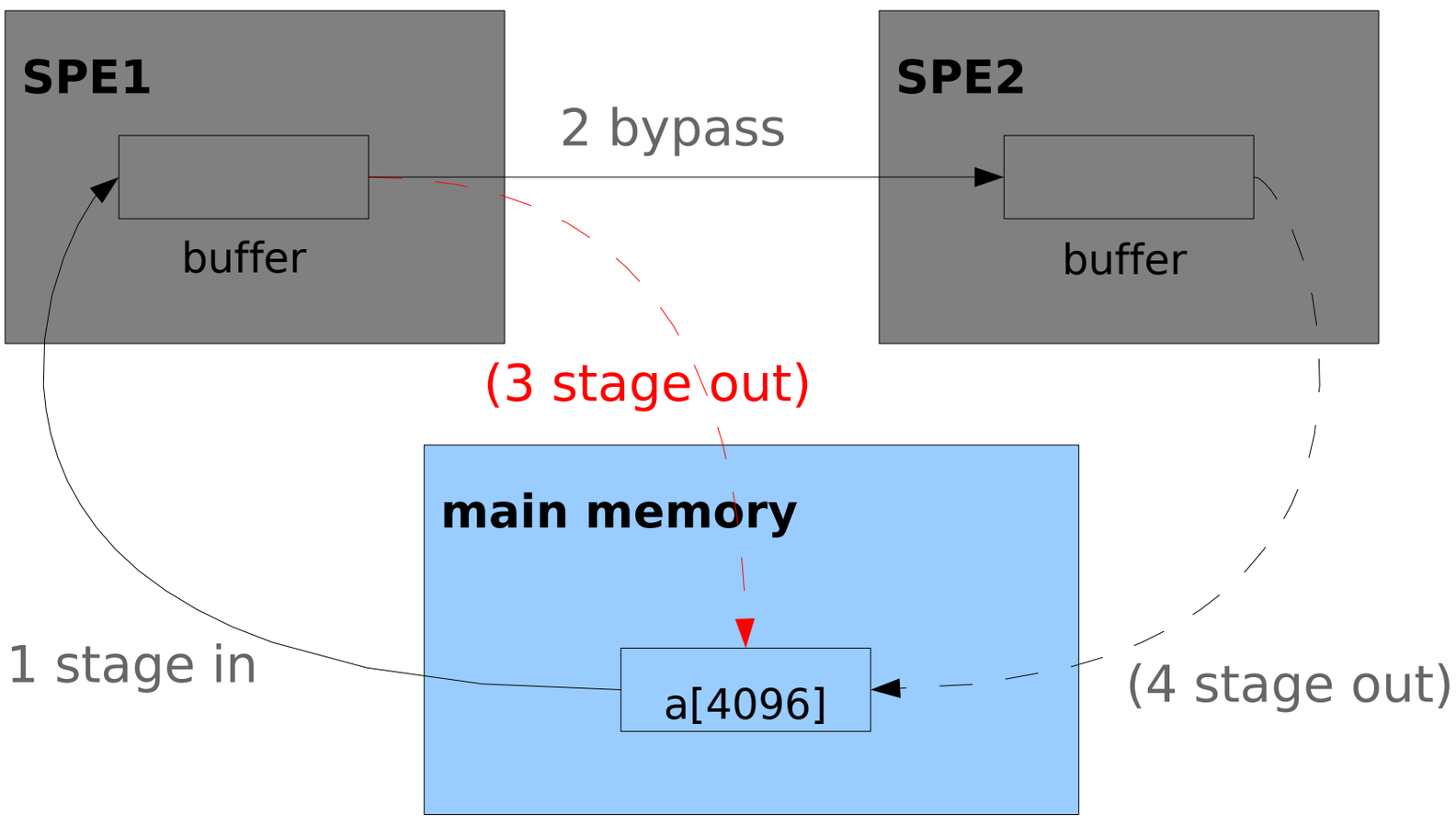






Lazy write-back: concept

- Do not tranfer objects back to main memory unless strictly necessary
- Exploit the information available in the bypassing mechanism
  - object versions
  - read count of a version
- Token passing to avoid early stage-outs

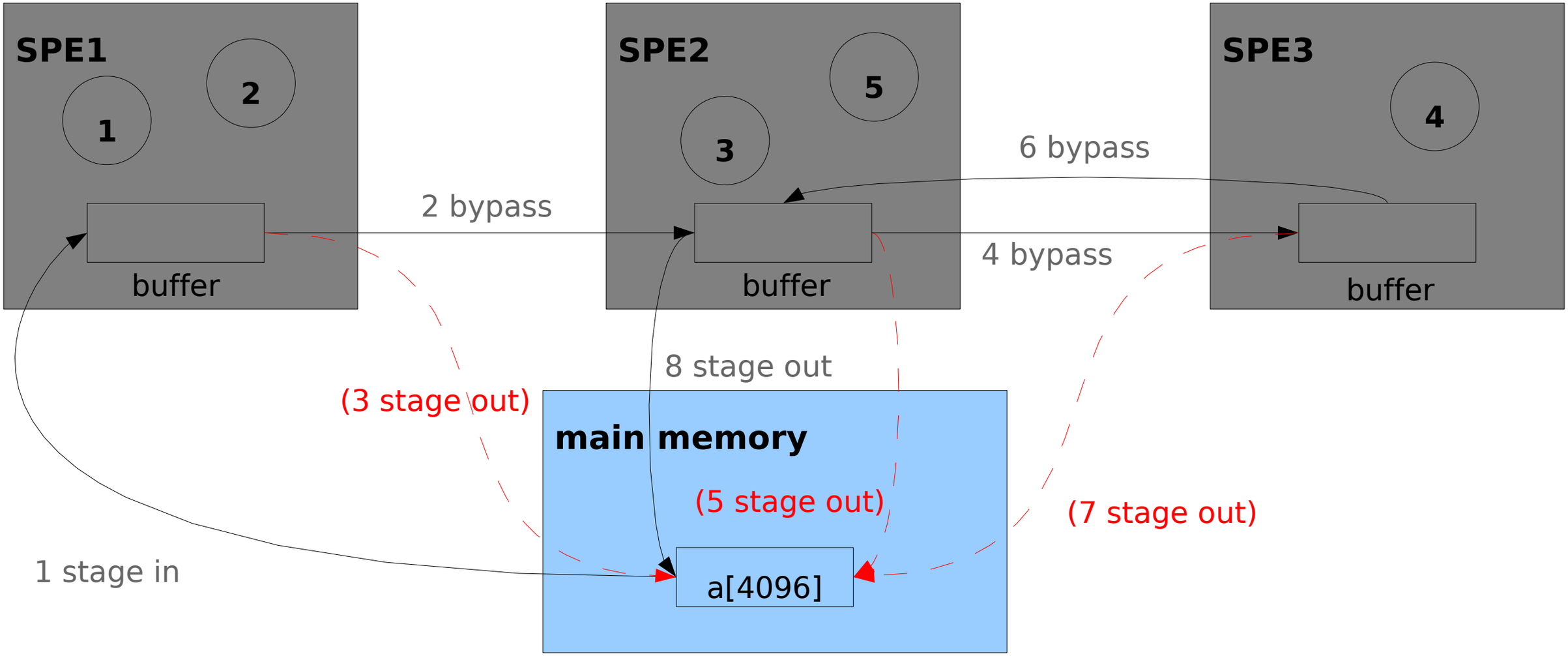



```
#pragma css task inout(a)
void foo(int a[4096]);
int a[4096];

int main(int argc, char *argv[])
{
...
foo(a);
...
foo(a);
...
return 0;
}
```

Lazy write-back: example

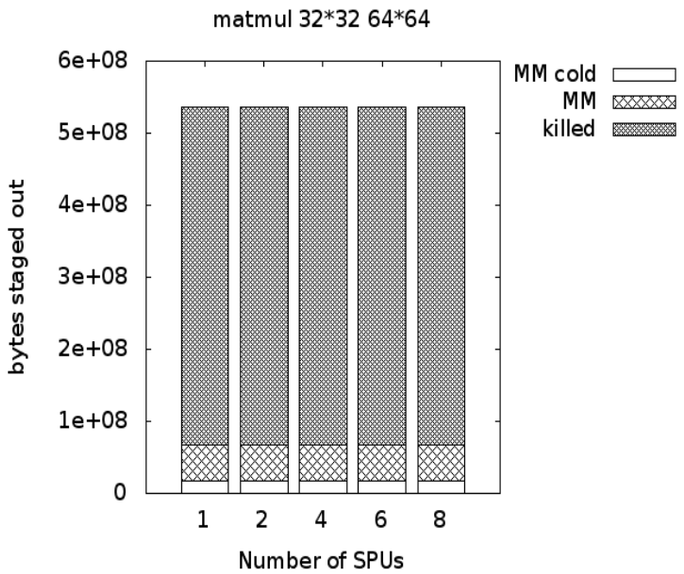
1. Task 1 reads and writes a → Obj(a,1)
2. Task 2 reads a (Obj(a,1))
3. Task 3 reads a (Obj(a,1))
4. Task 4 reads and writes a → Obj(a,2)
5. Task 5 reads a (Obj(a,2))

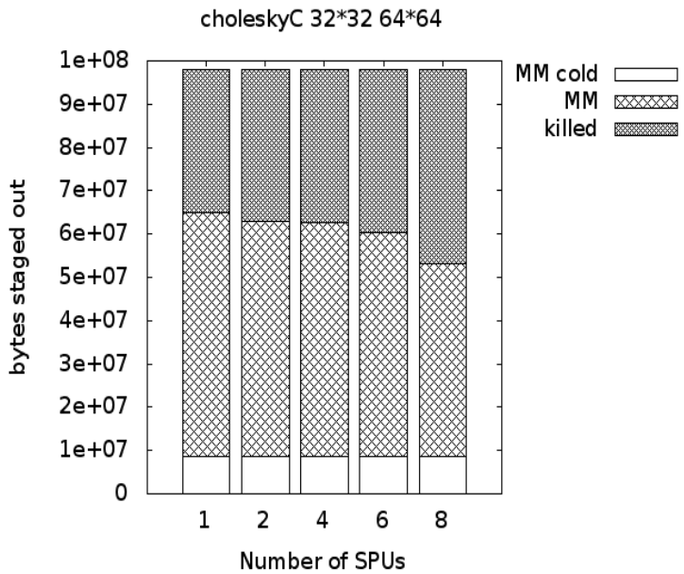
- Below is the perfect scenario
- Variations possible depending on relative ordering of execution of tasks and schedule

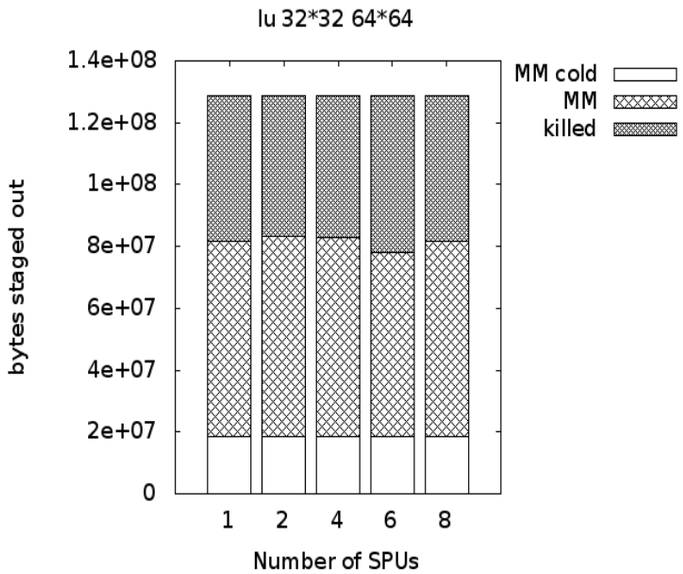


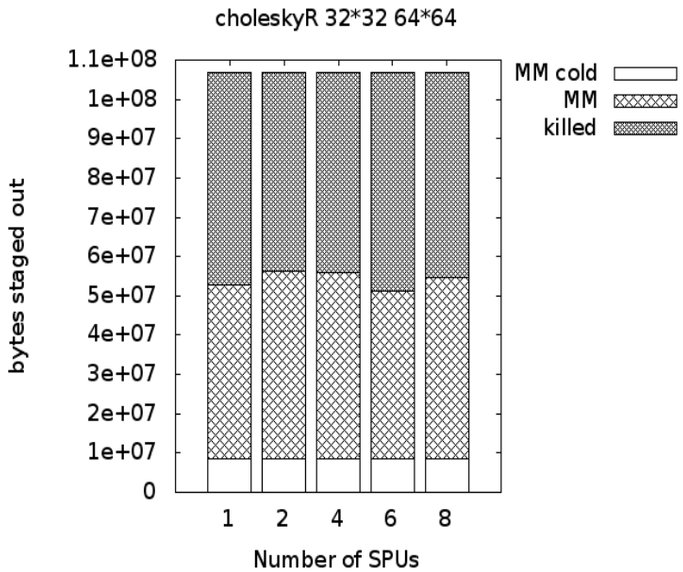
Lazy write-back: results

Can we avoid a significant fraction of the tranfers to main memory?



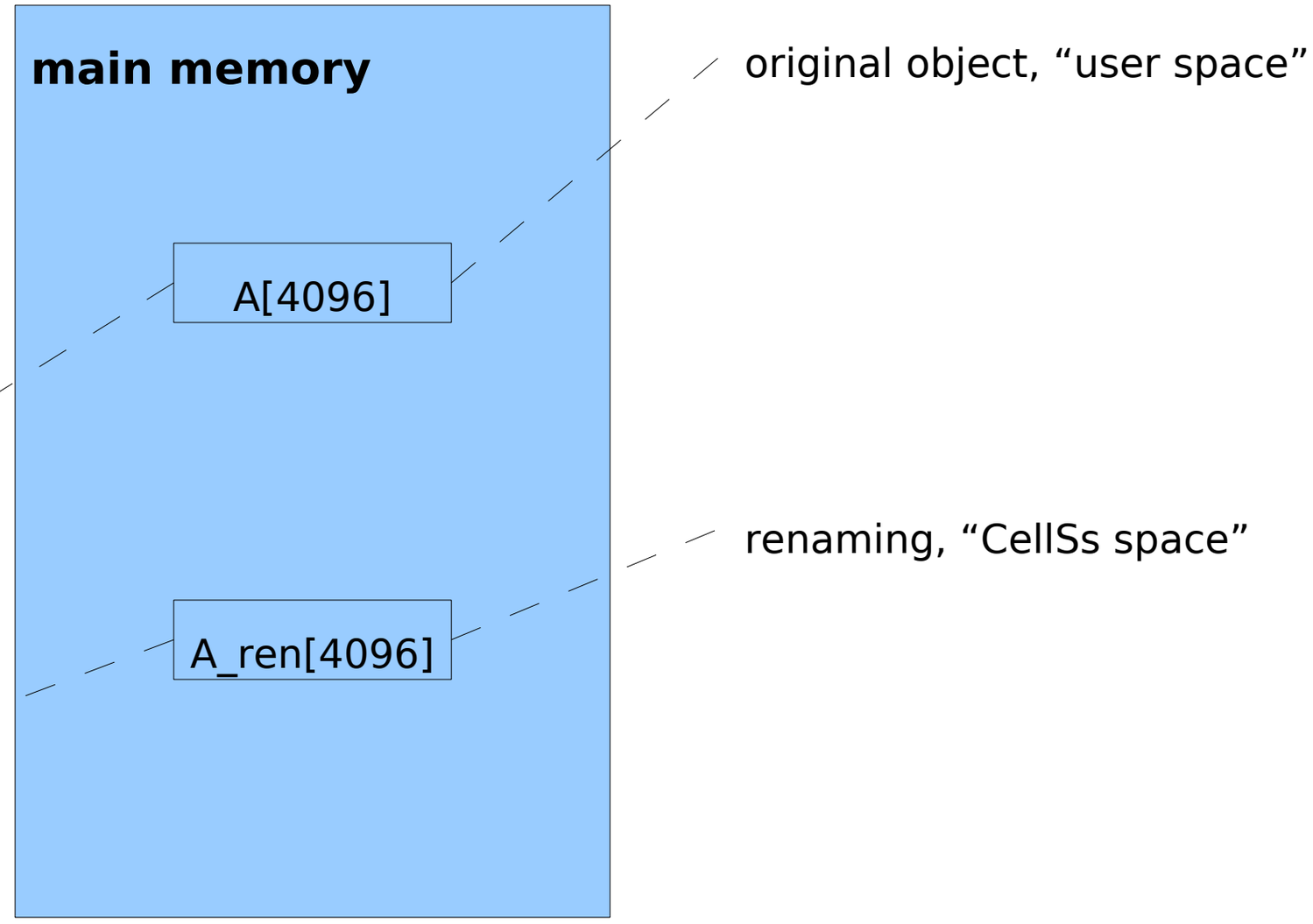






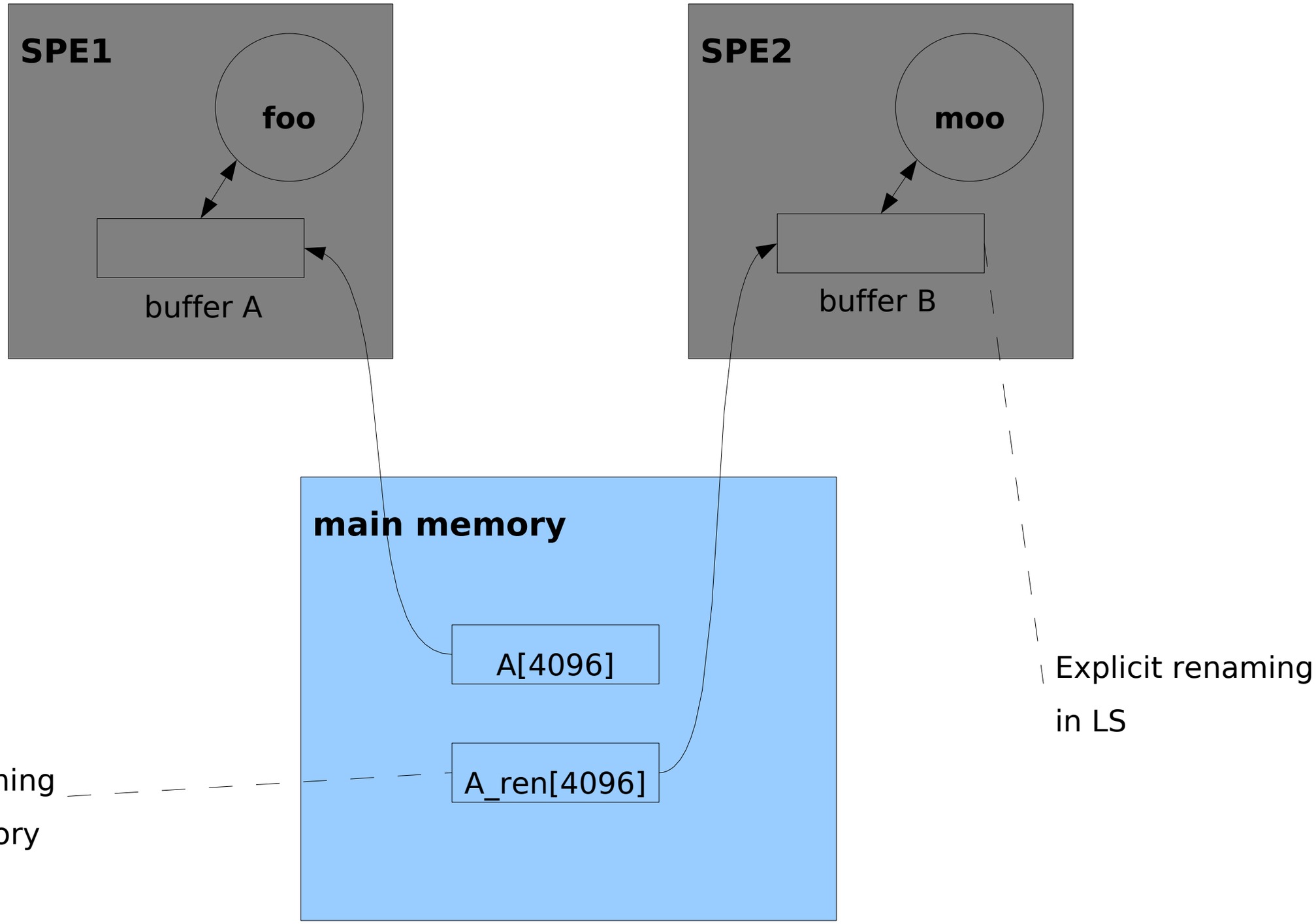
Renaming: traditional concept

```
#pragma css task inout(a)
void foo(int a[4096]);

#pragma css task out(a)
void moo(int a[4096]);

int a[4096];

int main(int argc, char *argv[])
{
...
foo(a)
...
moo(a)
...
return 0;
}
```

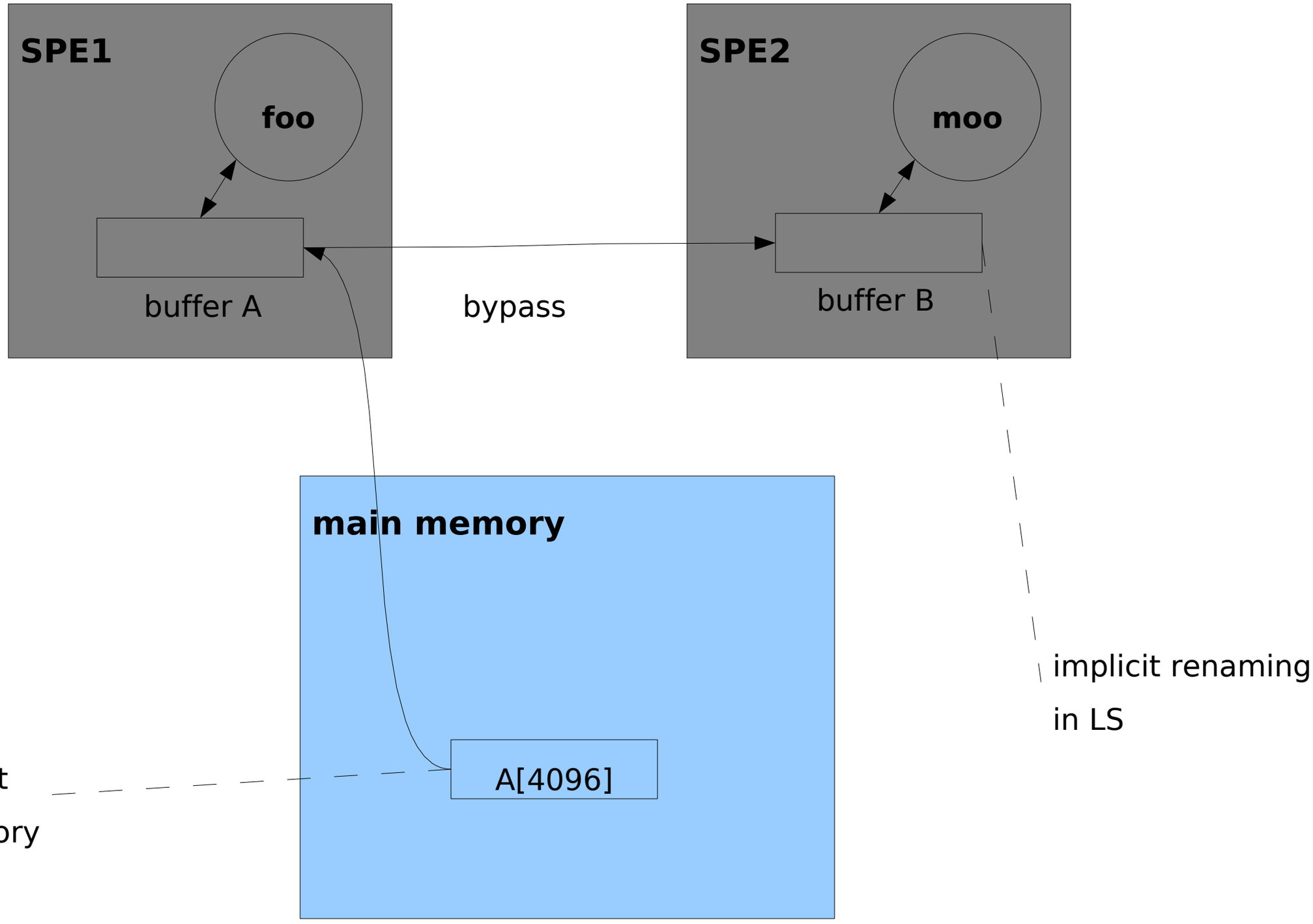


- Renaming improves parallelism at the cost of extra memory.
- Centralized

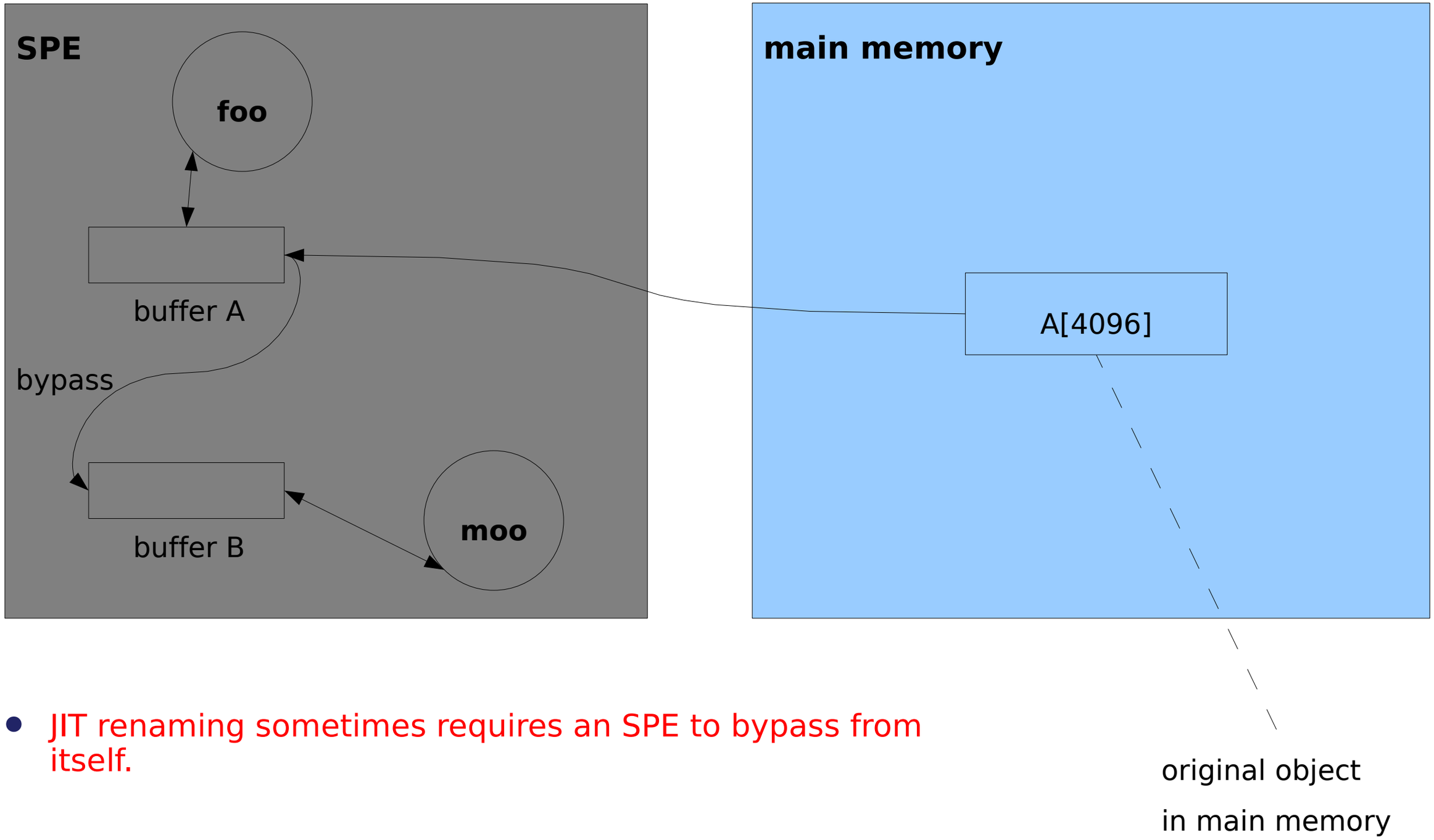
Renaming: traditional concept



Renaming: JIT renaming

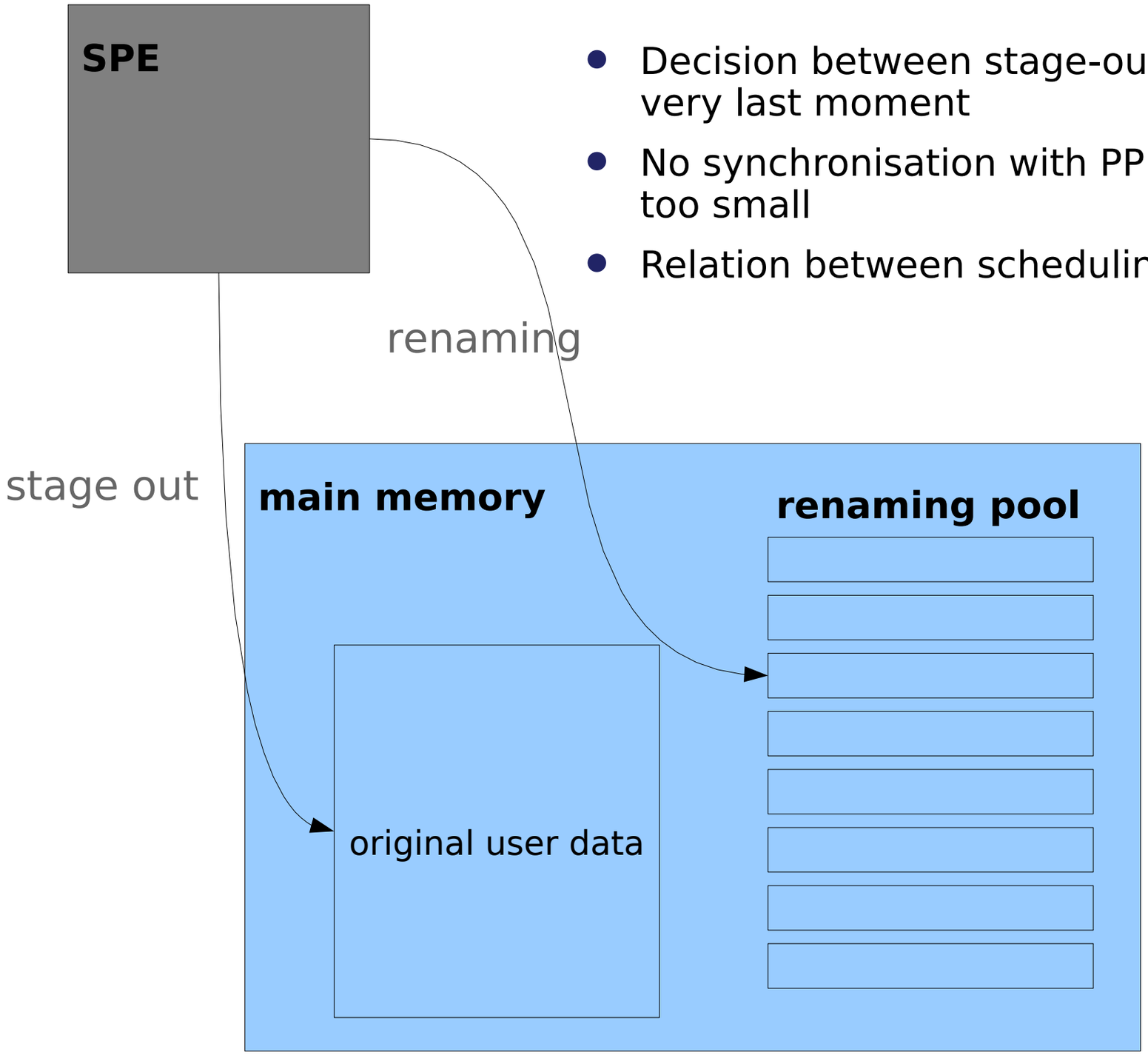


Renaming: JIT renaming



- JIT renaming sometimes requires an SPE to bypass from itself.

Renaming: JIT renaming



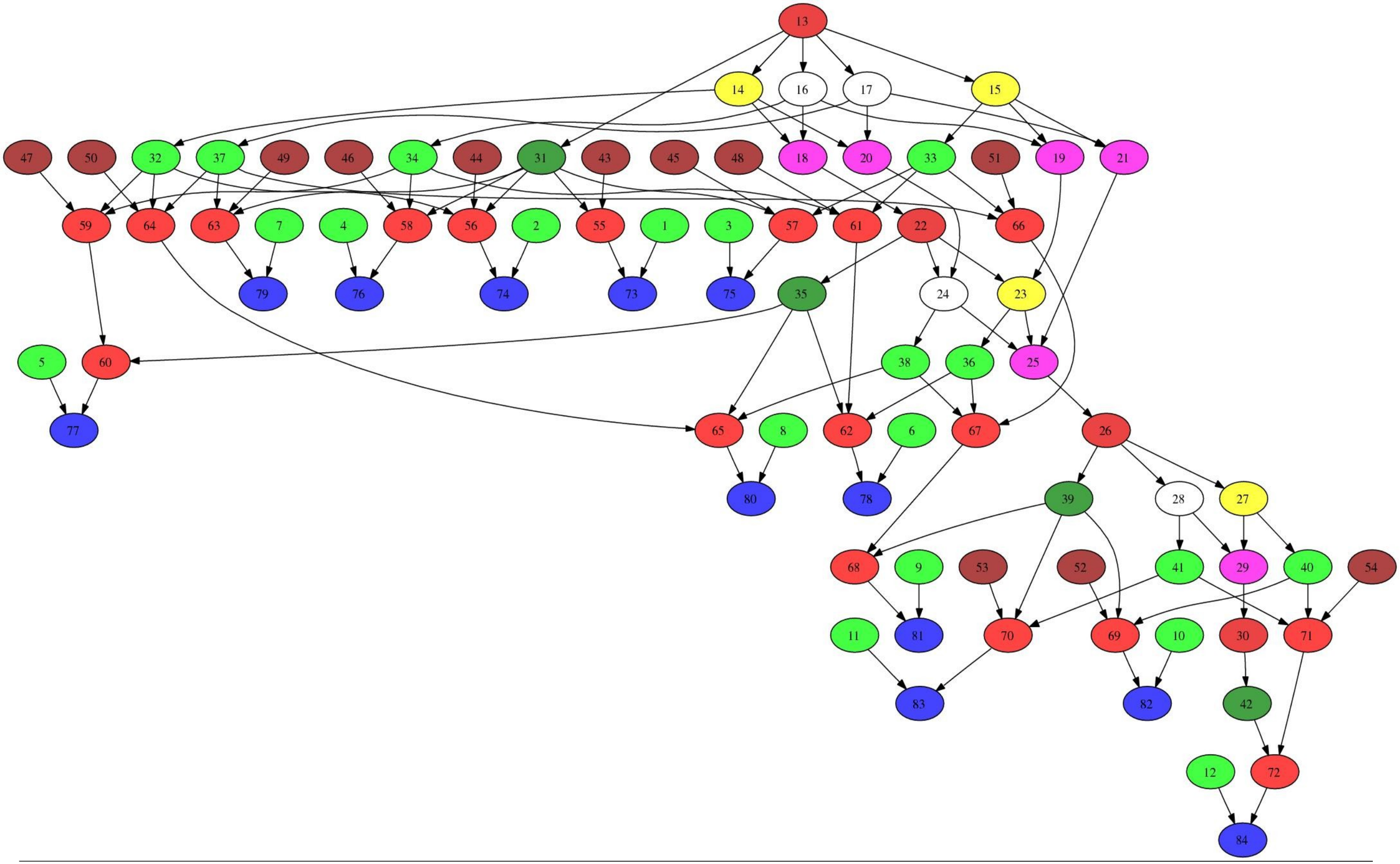
- Decision between stage-out or renaming made at the very last moment
- No synchronisation with PPE unless renaming pool too small
- Relation between scheduling and renaming

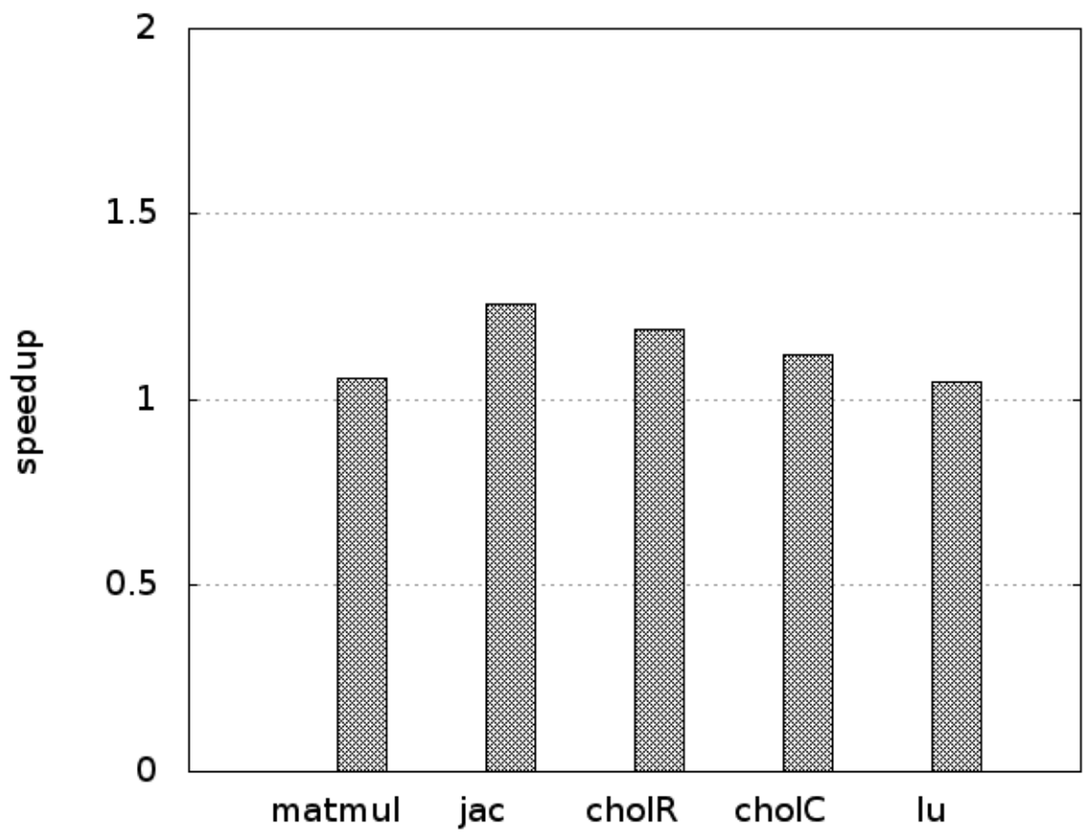
Ongoing work

- verification of the bypassing protocol
- studying ways to incorporate scheduling
  - distributed scheduling
  - shared representation of the Task Dependence Graph (TDG)

Questions?

task dependence graph (TDG)



Speedup results

- Very much work in progress
- Linear algebra applications on 16x16 hypermatrices of 64x64 floats
- Matrix multiplication, 2 variants of the Cholesky decomposition, a Jacobi computation and an LU decomposition.