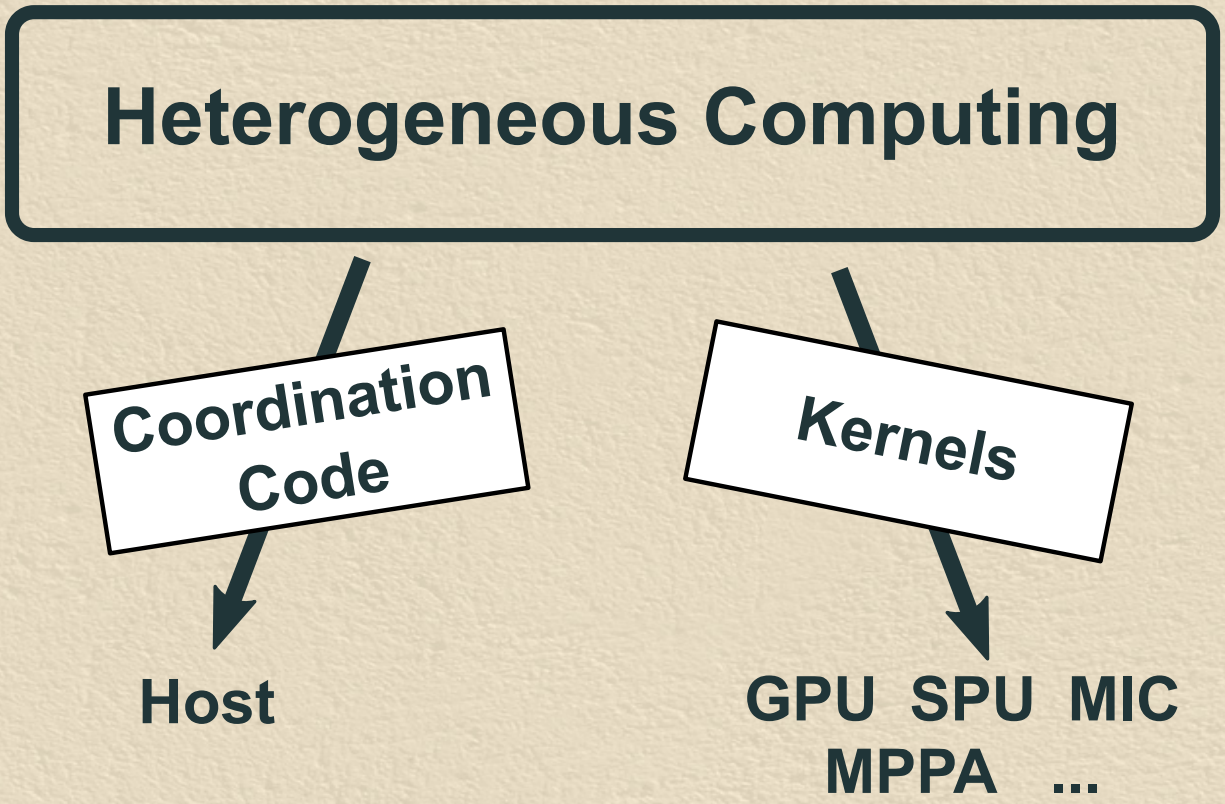
ViperVM: a runtime system for parallel functional high-performance computing on heterogeneous architectures

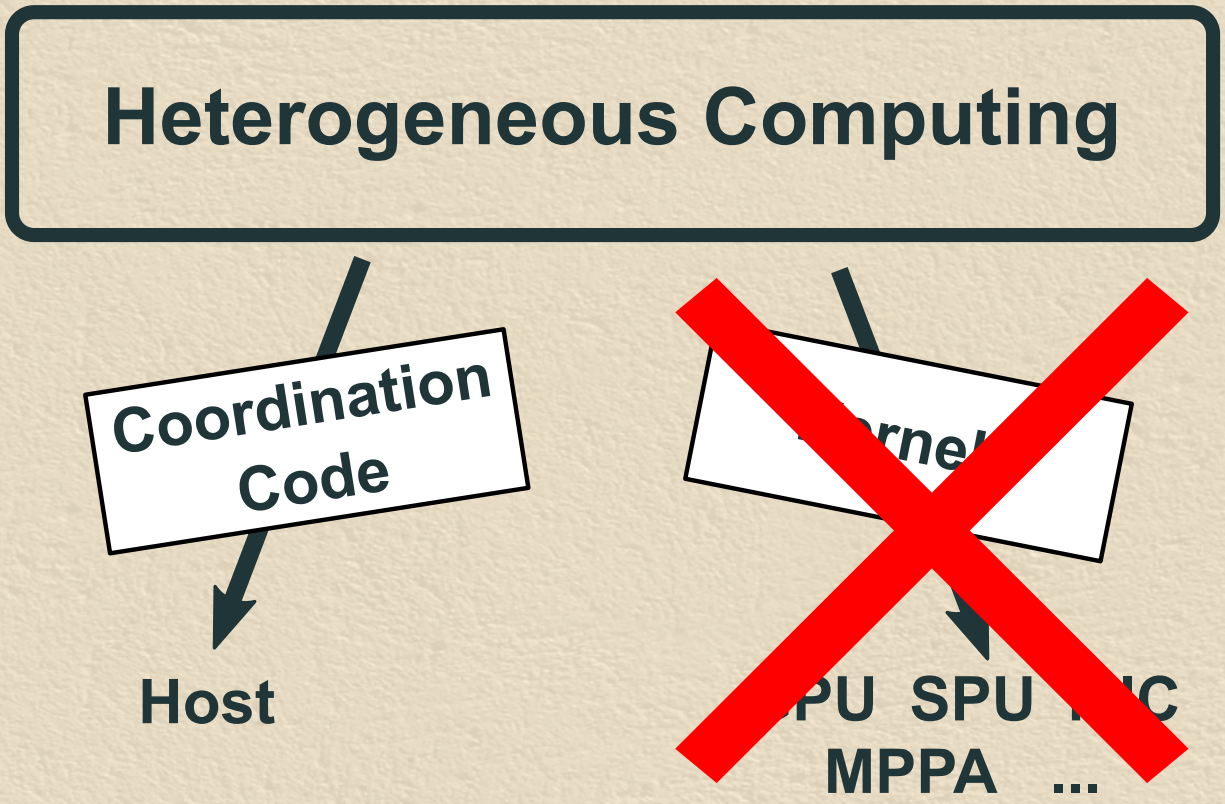
Sylvain Henry
sylvain.henry@inria.fr

FHPC'13 - Boston
September 23, 2013

What this talk is not about



What this talk is not about



Host code

Orchestrates

- ➔ Buffer allocations/releases in all memories
- ➔ Data transfers between memories
- ➔ Kernel compilations for all devices
- ➔ Kernel executions for all devices
- ➔ ...

Host code

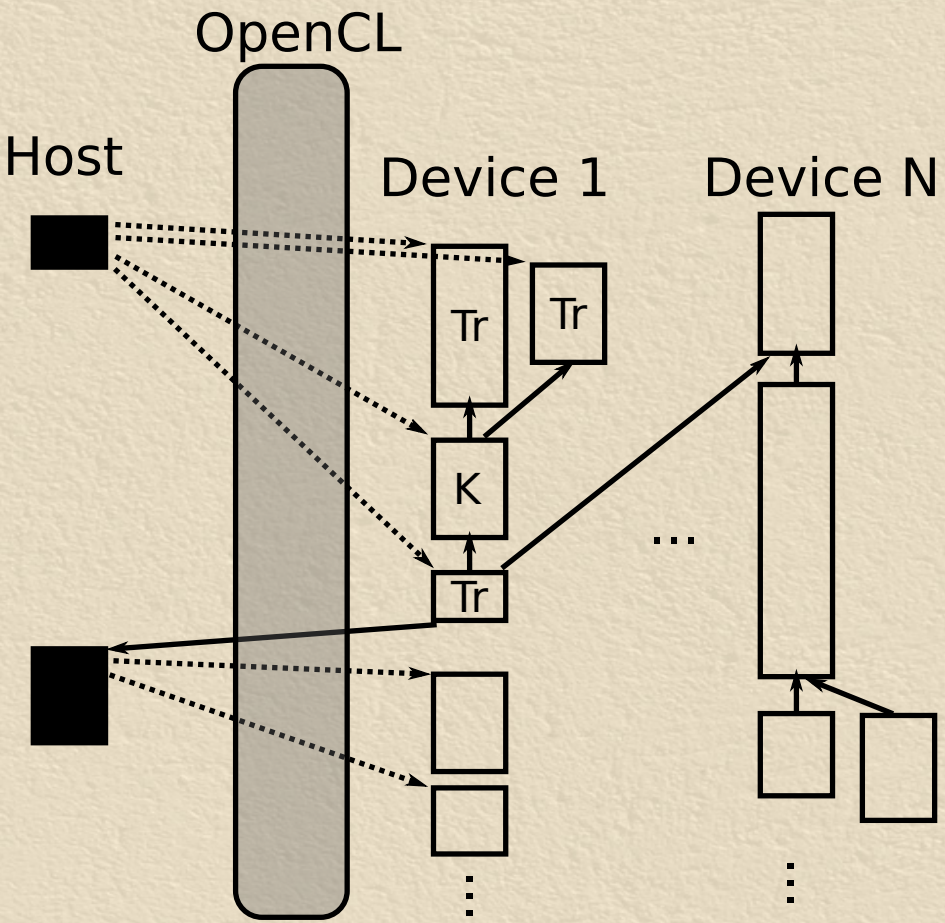
Orchestrates

- ➔ Buffer allocations/releases in all memories
- ➔ Data transfers between memories
- ➔ Kernel compilations for all devices
- ➔ Kernel executions for all devices
- ➔ ...

Most of these actions are performed asynchronously

Dynamic Command Graph

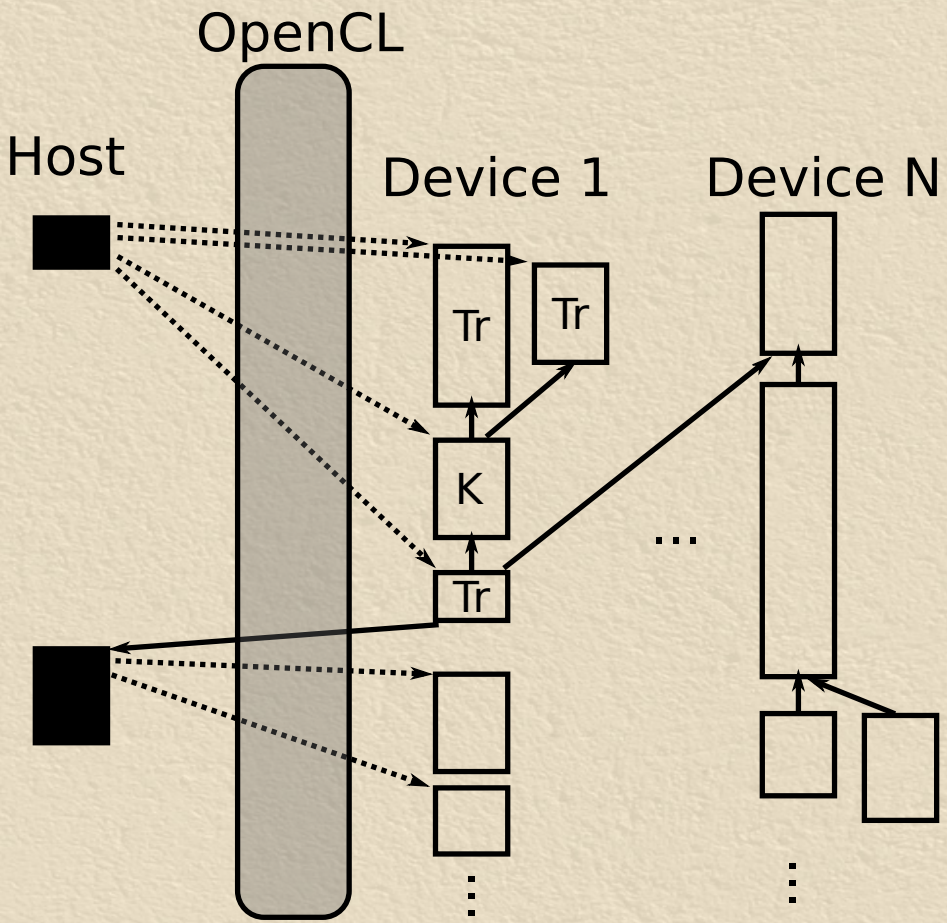
e.g. OpenCL, CUDA...



- ➔ Dynamic command graph creation
- ➔ Callback driven approach

Dynamic Command Graph

e.g. OpenCL, CUDA...

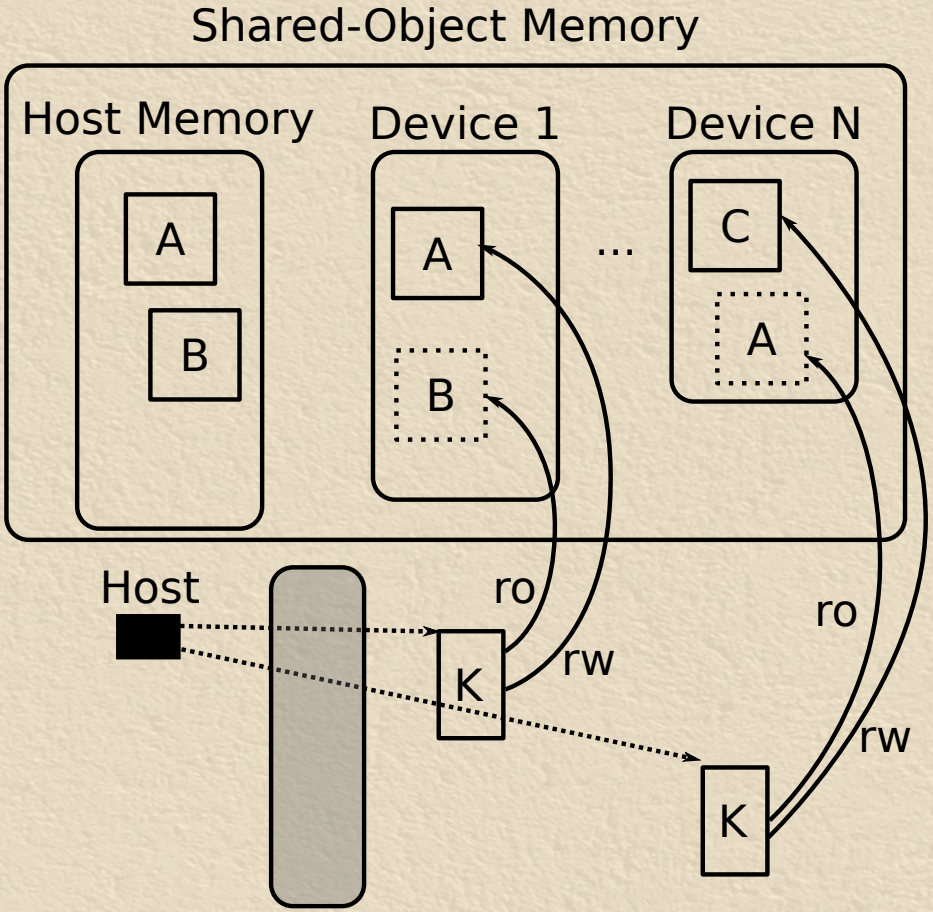


- ➔ Dynamic command graph creation
- ➔ Callback driven approach
- ➔ **Very low-level approach**

Can we do better?

Shared-Object Memory

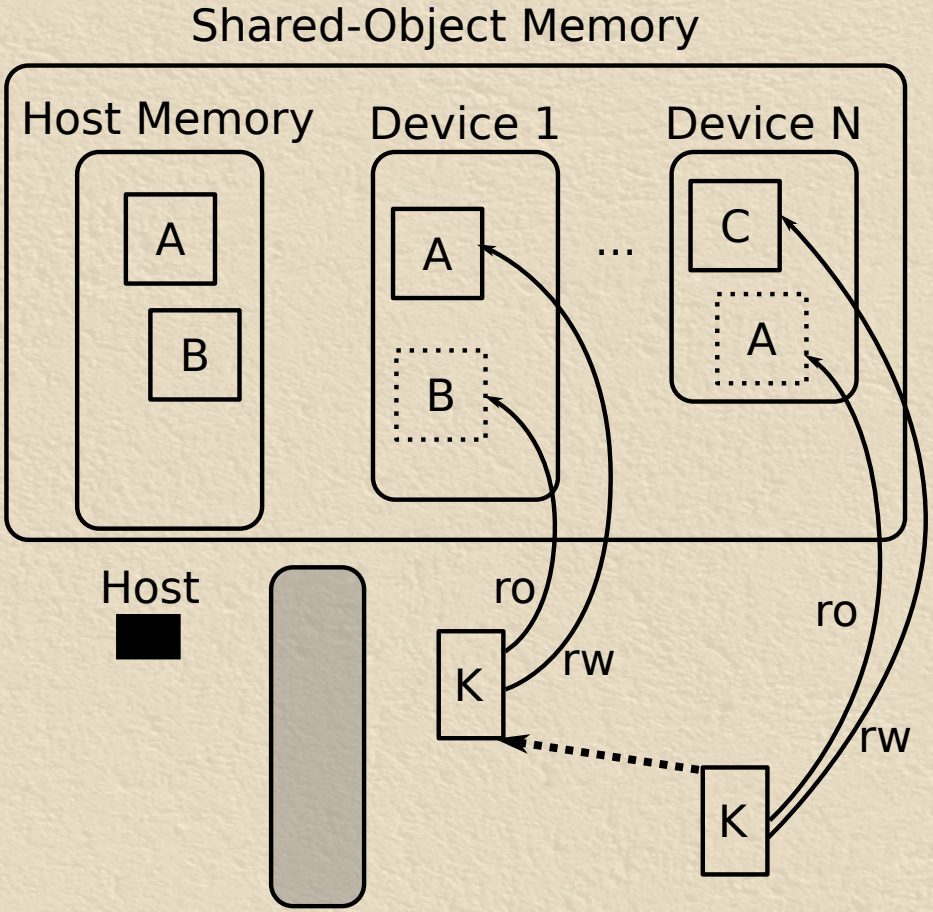
e.g. OpenACC, OpenHMPP... (to some extent)



➔ Kernel data access modes (RO,RW) specified

Shared-Object Memory

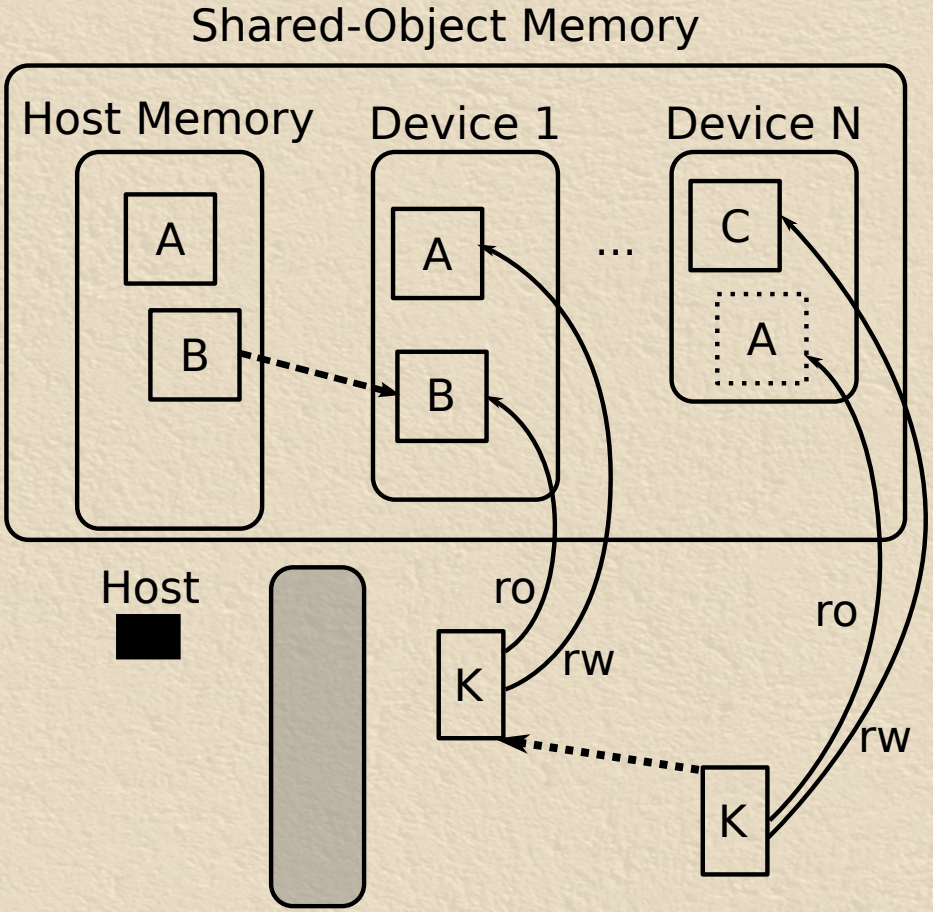
e.g. OpenACC, OpenHMPP... (to some extent)



- Kernel data access modes (RO,RW) specified
- Implicit read-after-write and write-after-write dependencies

Shared-Object Memory

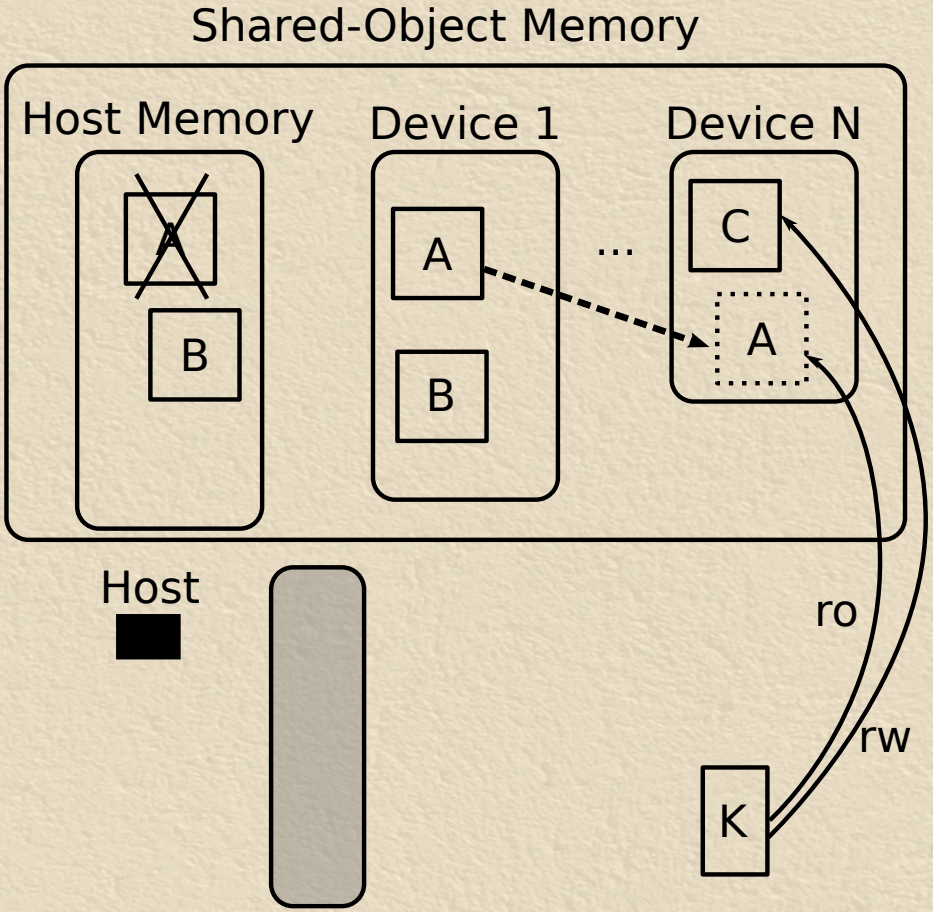
e.g. OpenACC, OpenHMPP... (to some extent)



- ➔ Kernel data access modes (RO,RW) specified
- ➔ Implicit read-after-write and write-after-write dependencies
- ➔ Implicit data transfers

Shared-Object Memory

e.g. OpenACC, OpenHMPP... (to some extent)



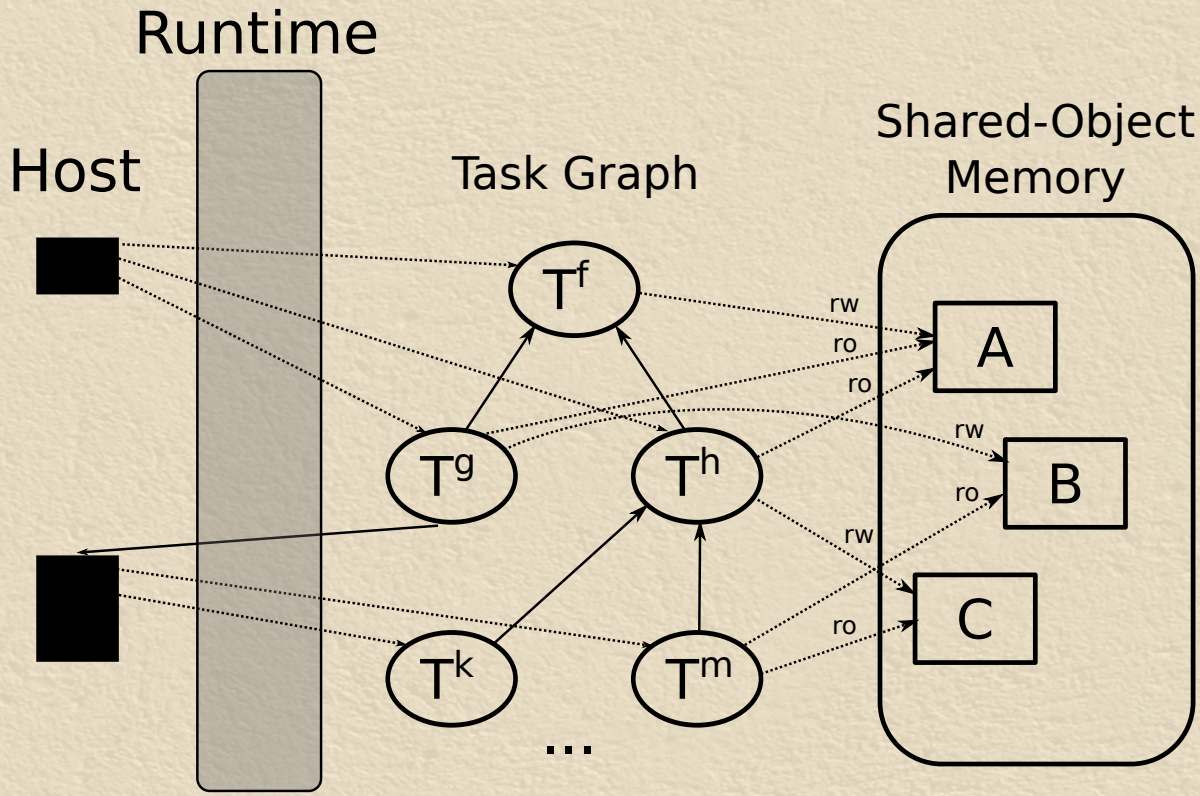
- ➔ Kernel data access modes (RO,RW) specified
- ➔ Implicit read-after-write and write-after-write dependencies
- ➔ Implicit data transfers
- ➔ Coherency ensured

Automatic data management

Can we do better?

Dynamic Task Graph

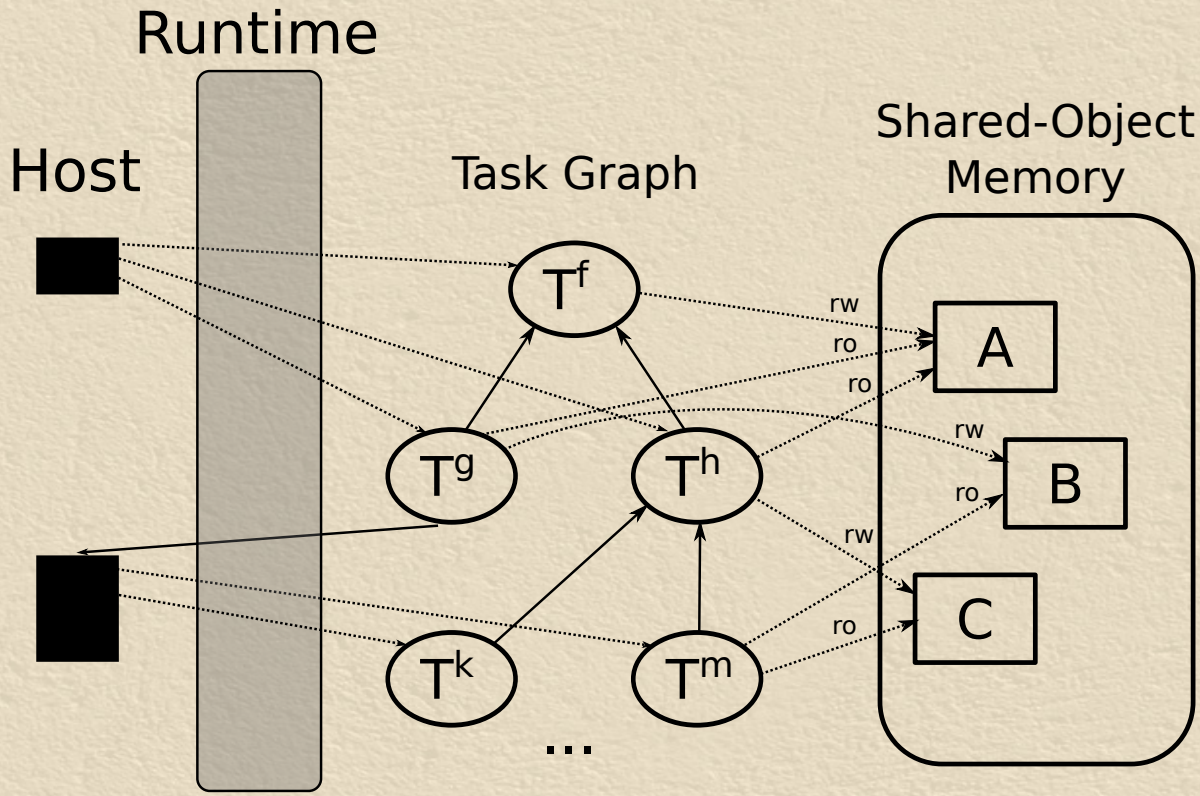
e.g. StarSS, StarPU...



- Task = Set of kernels
  - $T^f = \{K^f_{x86}, K^f_{OpenCL}, K^f_{CUDA} \cdots\}$

Dynamic Task Graph

e.g. StarSS, StarPU...



- Task = Set of kernels
  - $T^f = \{K^f_{x86}, K^f_{OpenCL}, K^f_{CUDA} \cdots\}$

Automatic kernel scheduling

Automatic data management

+

Automatic kernel scheduling

Problem solved?

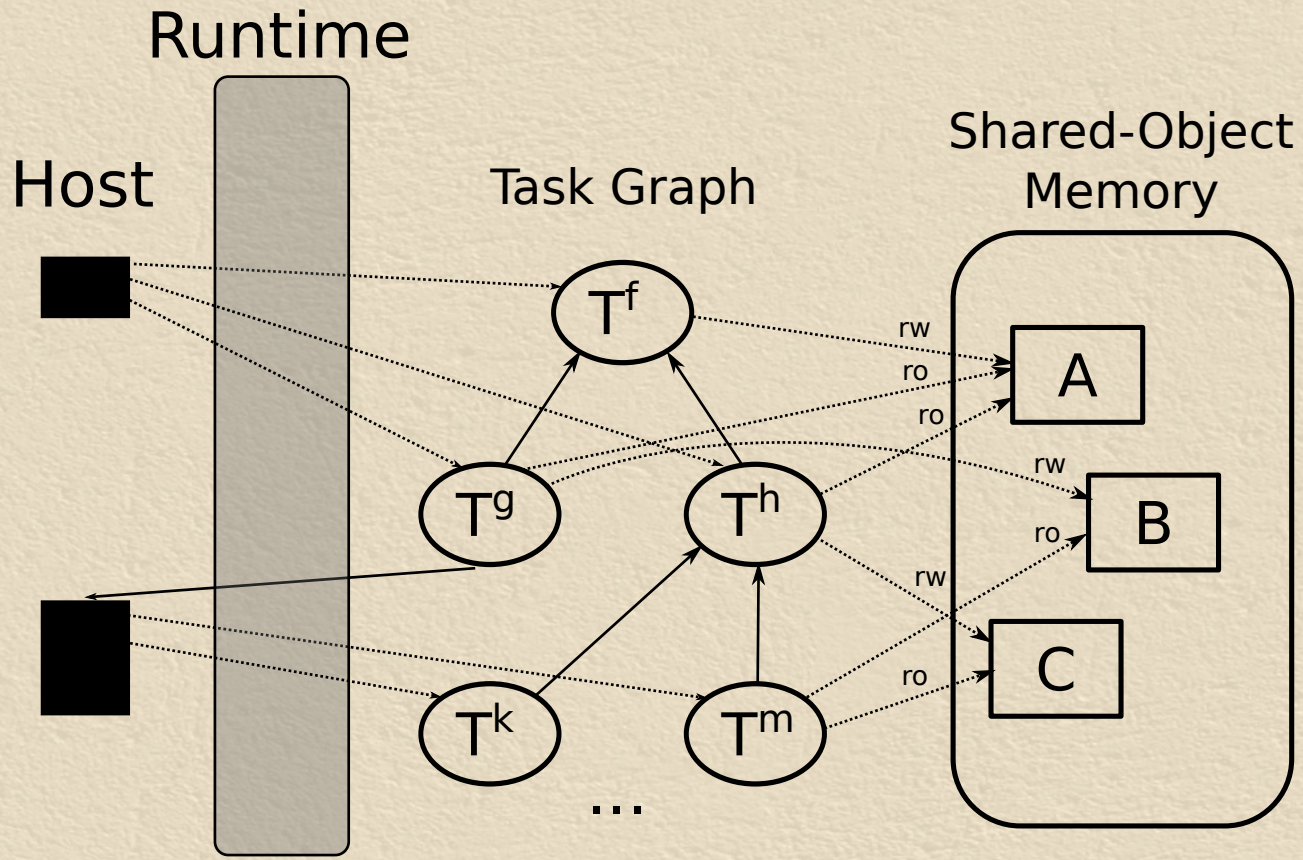
Automatic data management

$+$

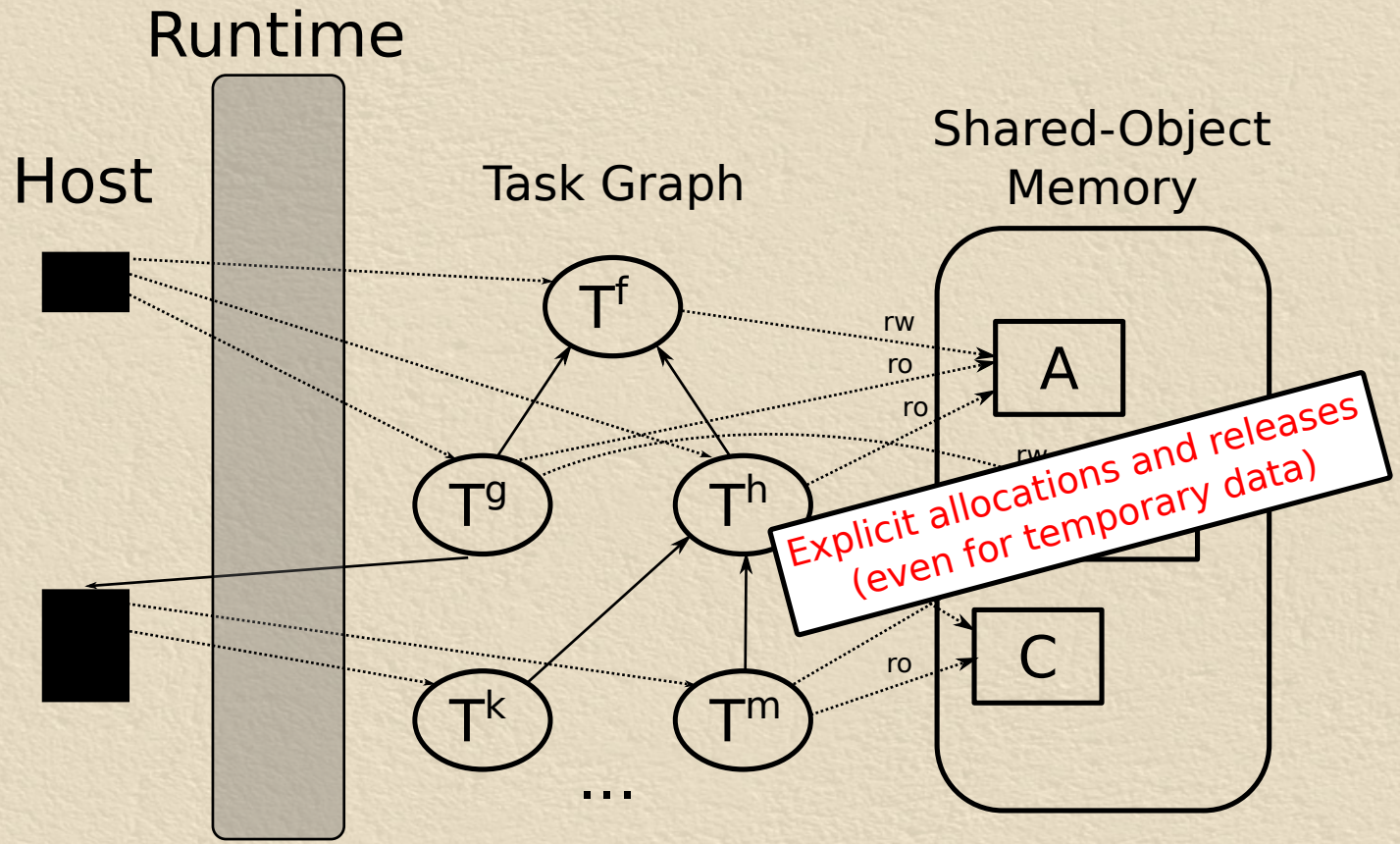
Automatic kernel scheduling

~~Problem solved?~~ Other issues

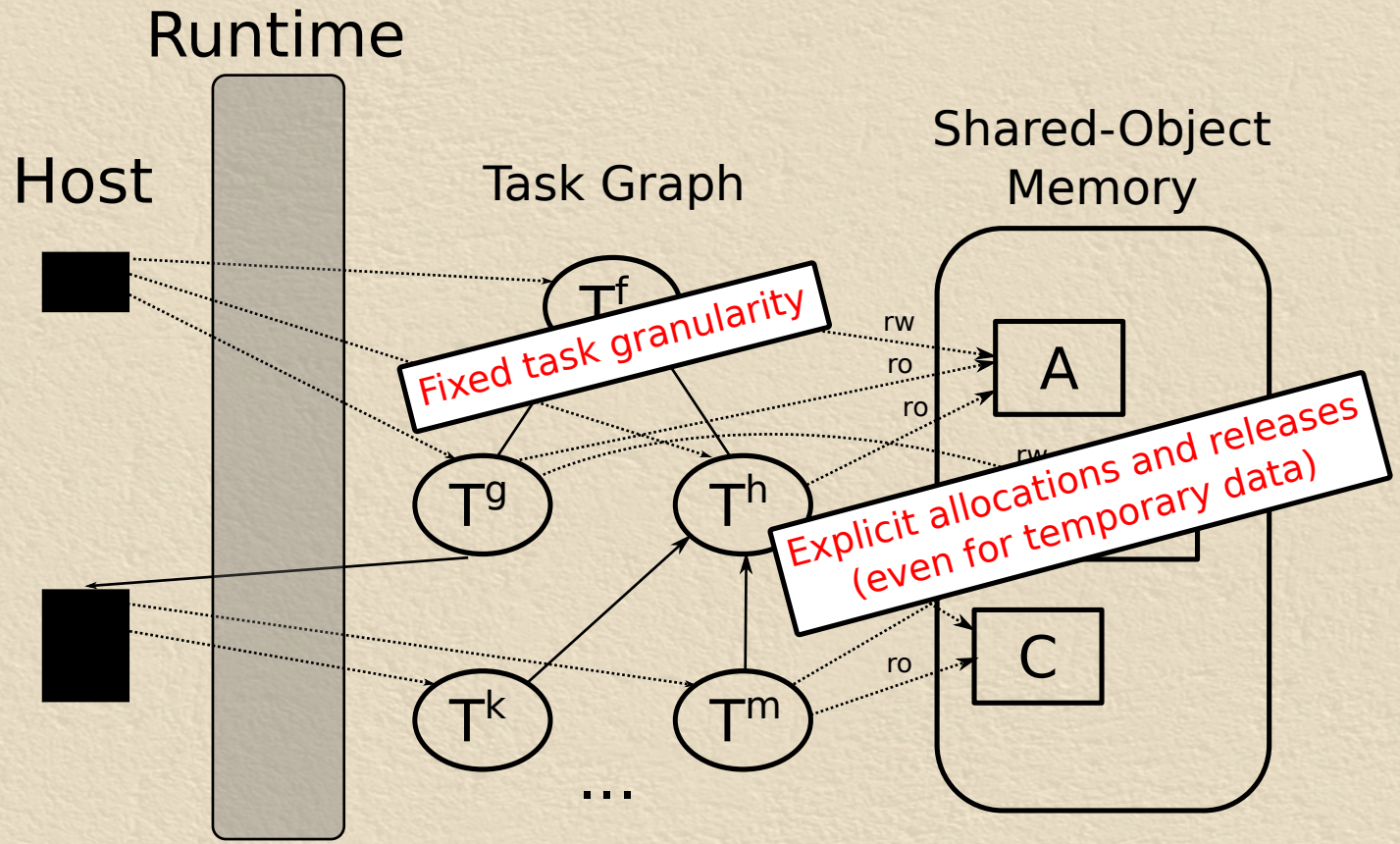
Dynamic Task Graph: Issues



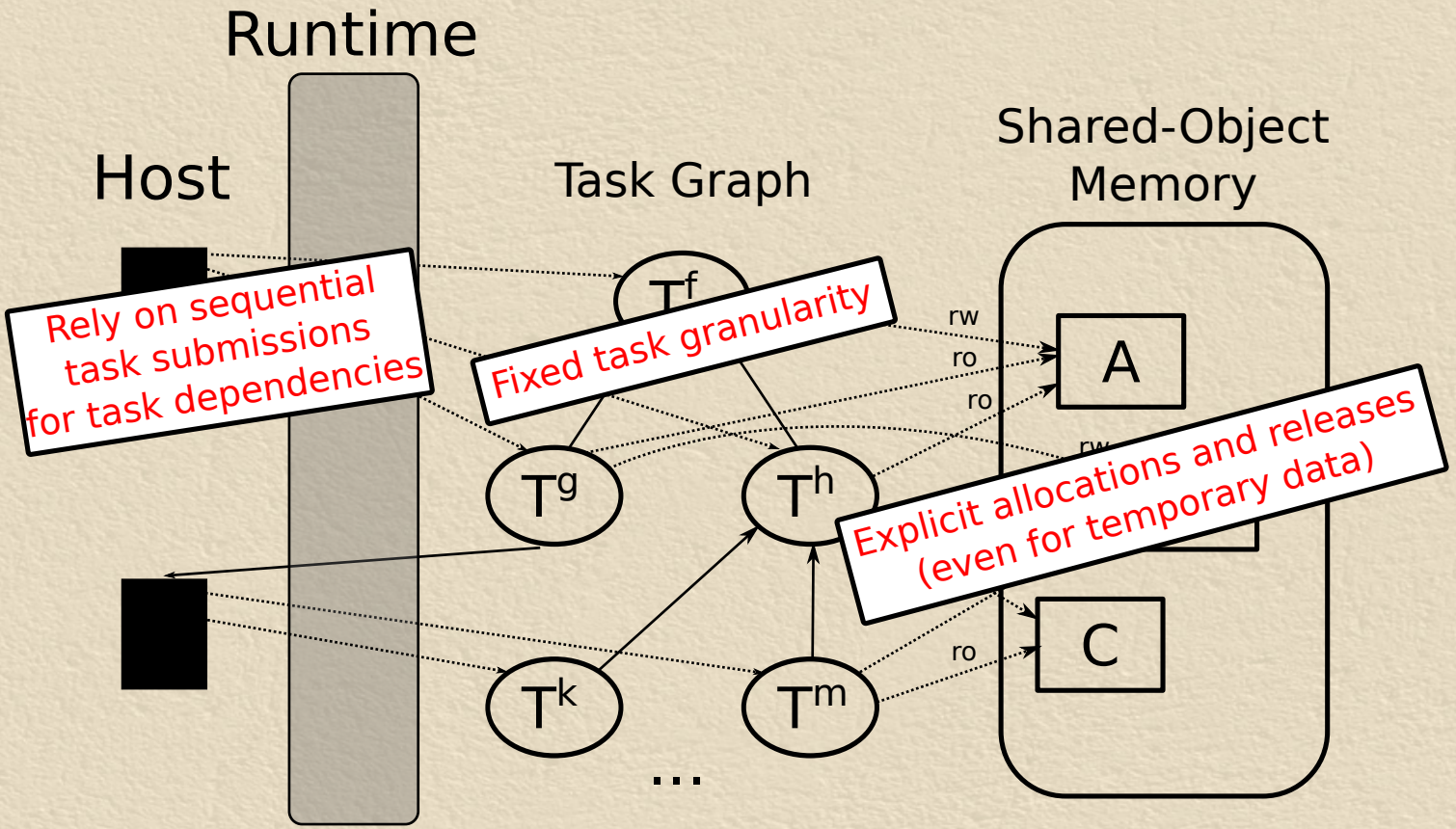
Dynamic Task Graph: Issues



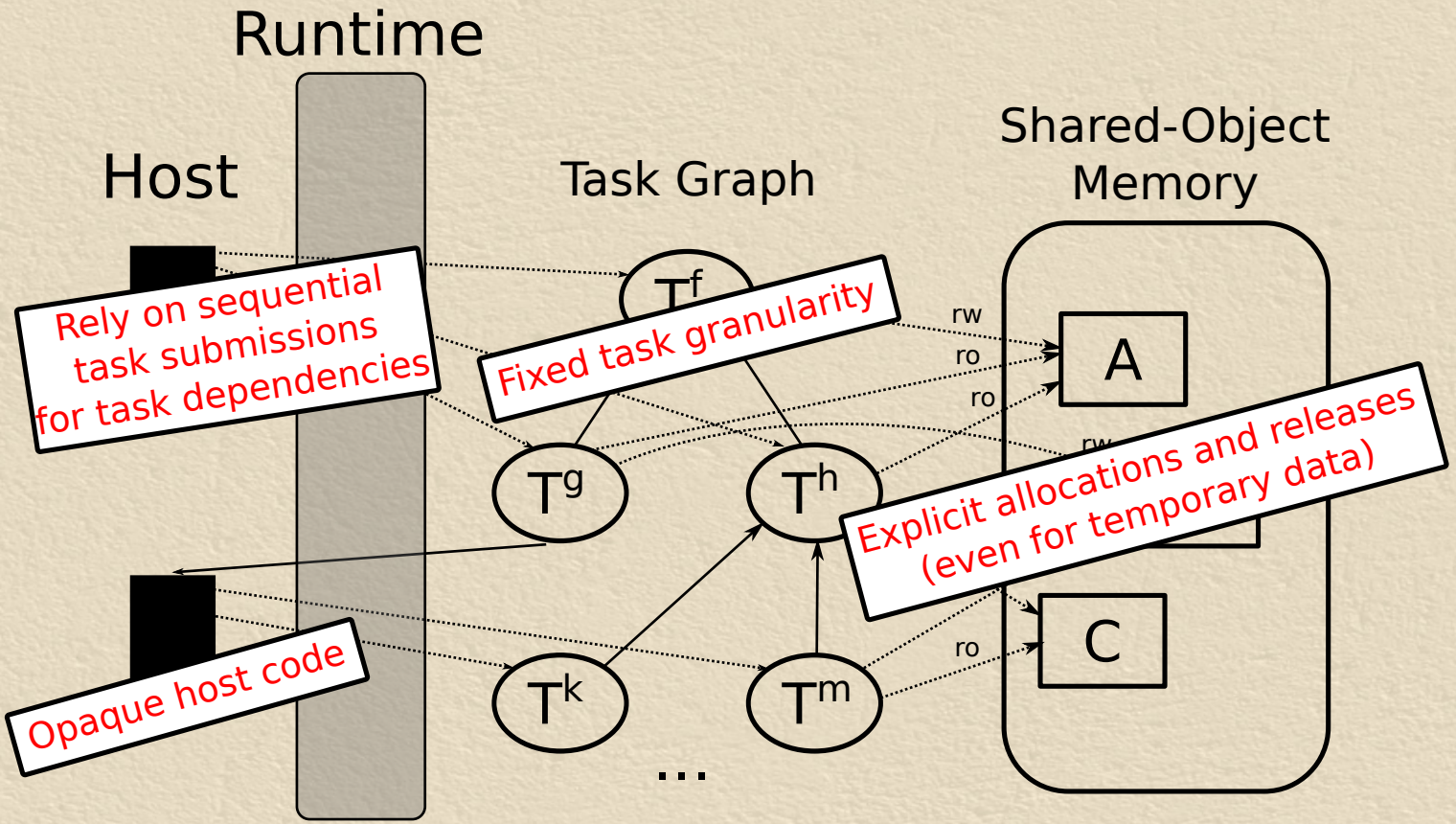
Dynamic Task Graph: Issues



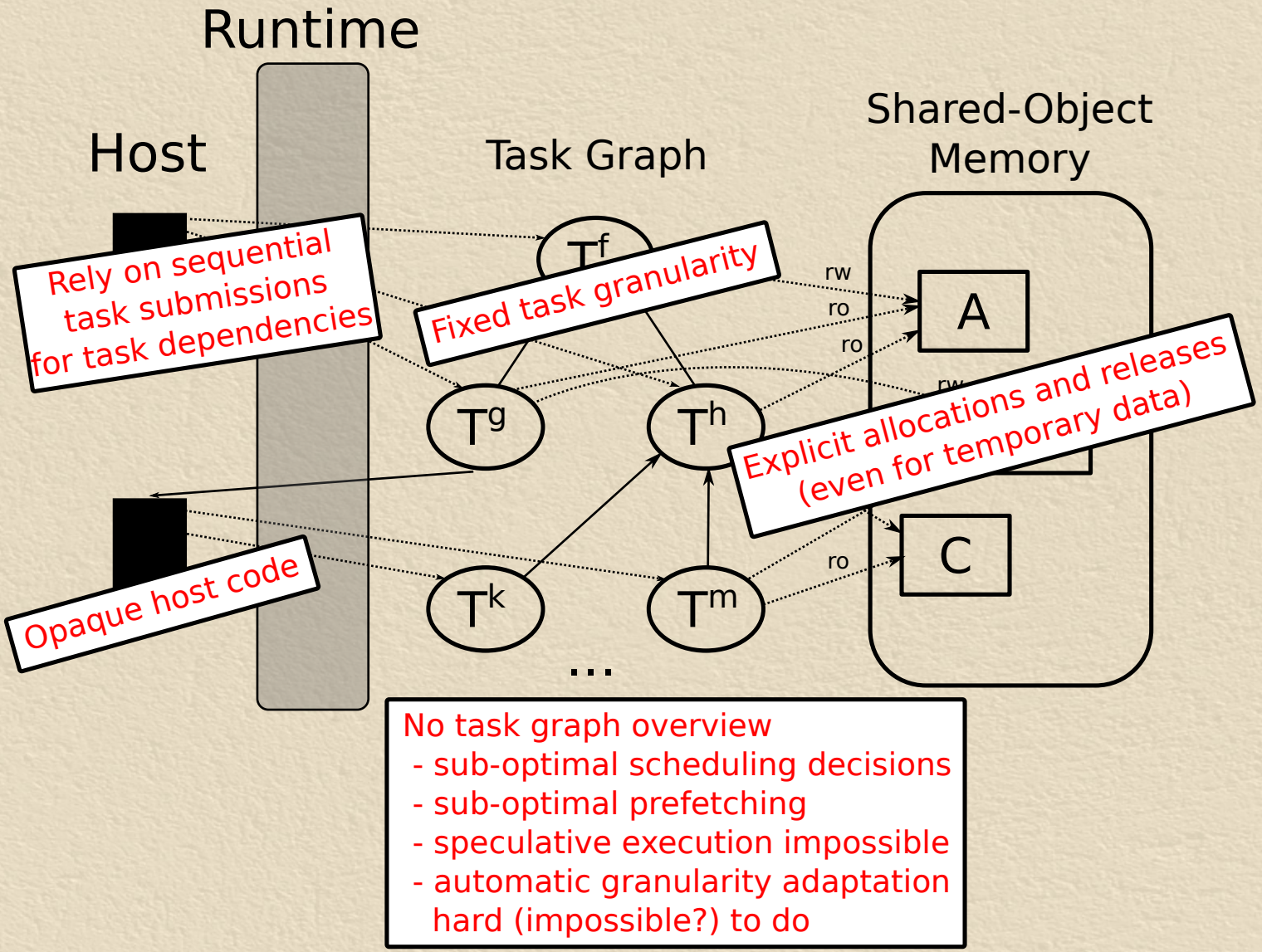
Dynamic Task Graph: Issues



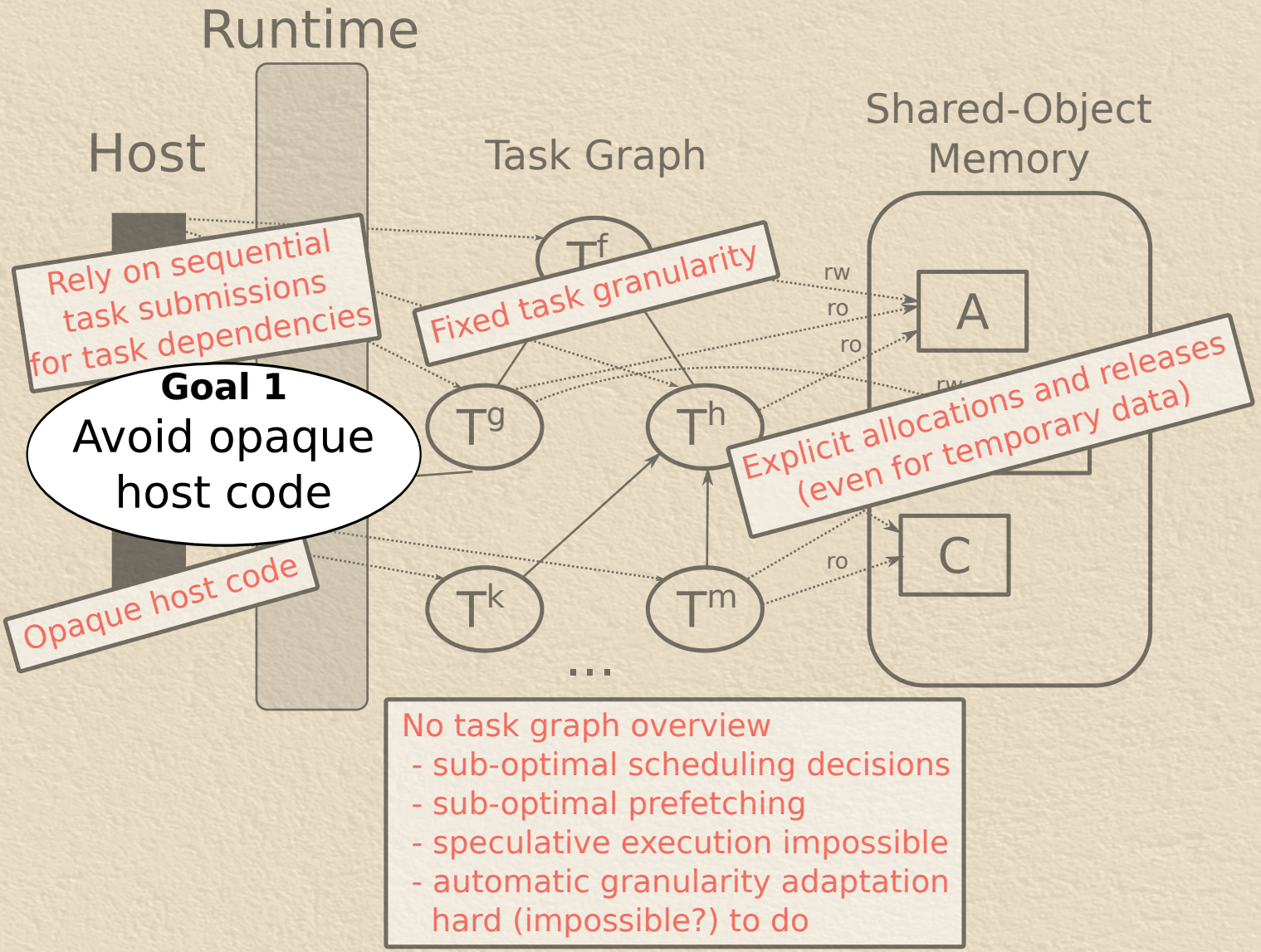
Dynamic Task Graph: Issues



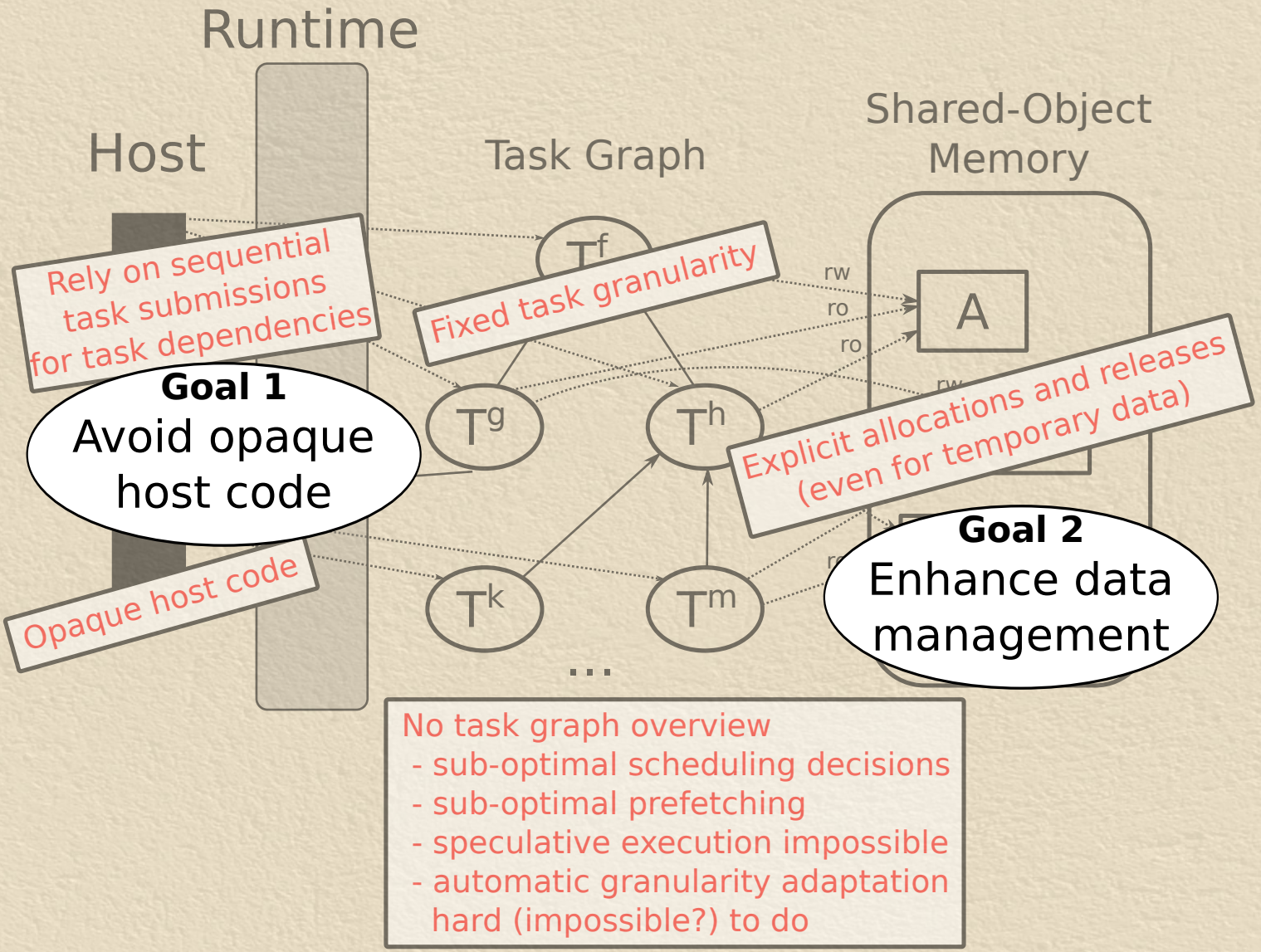
Dynamic Task Graph: Issues



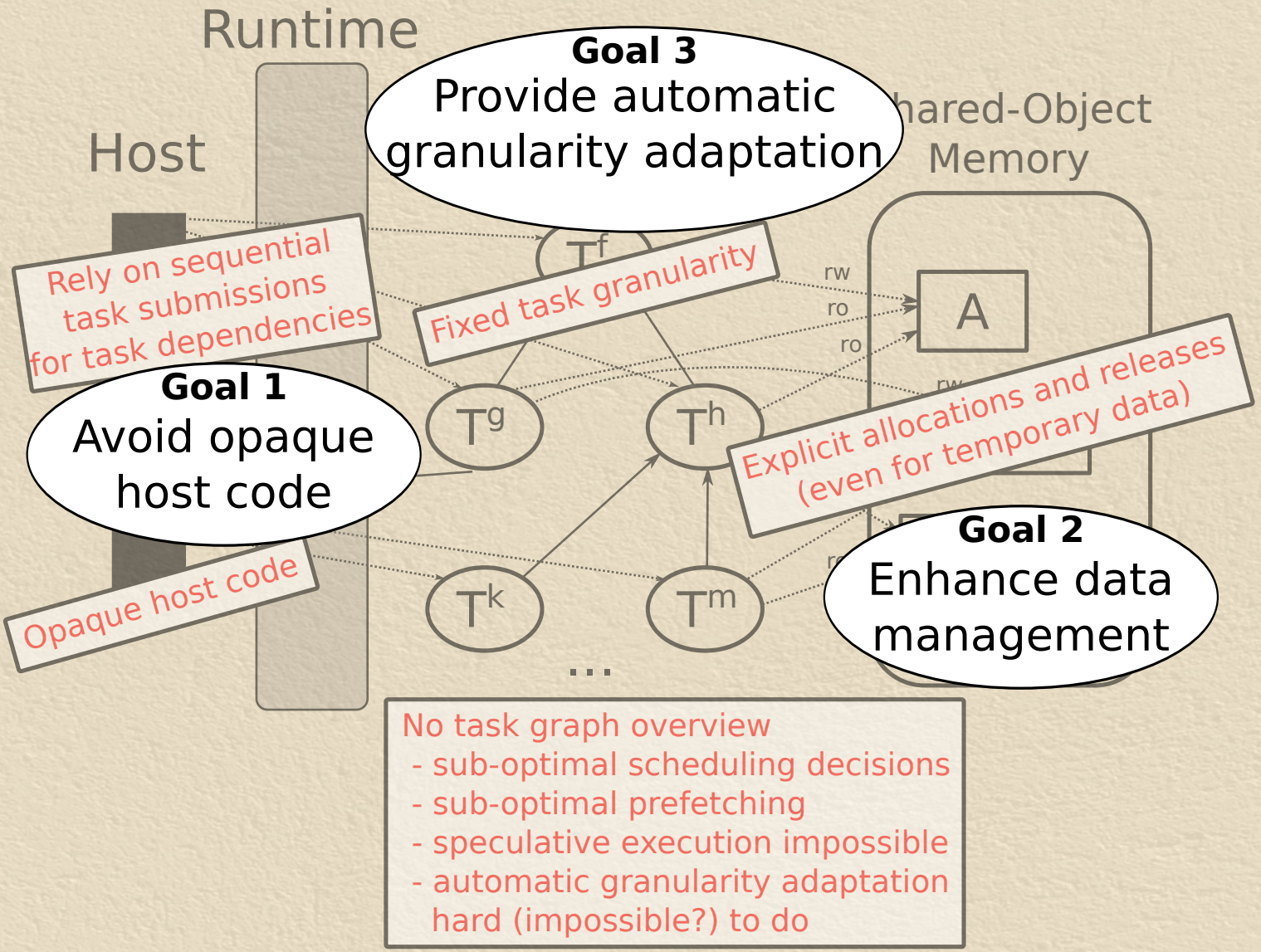
Challenges


Runtime
Host
Task Graph
Shared-Object Memory
Rely on sequential task submissions for task dependencies
Fixed task granularity
Goal 1
Avoid opaque host code
Opaque host code
Explicit allocations and releases (even for temporary data)
rw
ro
ro
rw
ro
A
C
T^f
T^g
T^h
T^k
T^m
...
No task graph overview
- sub-optimal scheduling decisions
- sub-optimal prefetching
- speculative execution impossible
- automatic granularity adaptation hard (impossible?) to do

Challenges


Runtime
Host
Task Graph
Shared-Object Memory
Rely on sequential task submissions for task dependencies
Fixed task granularity
Goal 1
Avoid opaque host code
Opaque host code
Explicit allocations and releases (even for temporary data)
Goal 2
Enhance data management
A
rw
ro
ro
rw
T^f
T^g
T^h
T^k
T^m
...
No task graph overview
- sub-optimal scheduling decisions
- sub-optimal prefetching
- speculative execution impossible
- automatic granularity adaptation hard (impossible?) to do

Challenges


Runtime
Host
Goal 3
Provide automatic granularity adaptation
hared-Object Memory
Rely on sequential task submissions for task dependencies
Fixed task granularity
Goal 1
Avoid opaque host code
Explicit allocations and releases (even for temporary data)
Goal 2
Enhance data management
Opaque host code
A
rw
ro
ro
rw
T^f
T^g
T^h
T^k
T^m
...
No task graph overview
- sub-optimal scheduling decisions
- sub-optimal prefetching
- speculative execution impossible
- automatic granularity adaptation hard (impossible?) to do

Idea

Use functional programs to describe task graphs

+

Parallel evaluation

Idea

Use functional programs to describe task graphs

+

Parallel evaluation

Why?

- ➔ Kernels (and tasks) $\simeq$ Built-in super-combinators
  - ➔ Functional programs are graphs of super-combinators
- ➔ Finding next task to execute $\simeq$ Finding next redex
- ➔ Data only accessed by their handles in both cases

Motivating example

Enhanced data management

Suppose we want to compute (a, b, c and d are matrices):

```
r = (a−b) * (c+d)
```

Motivating example

Enhanced data management

Suppose we want to compute (a, b, c and d are matrices):

```
r = (a-b) * (c+d)
```

Instead of (pseudo-code):

```
do
   tmp1 <- allocateMatrixData (width a) (height a)...
   tmp2 <- allocateMatrixData (width c) (height c)...
   r <- allocateMatrixData (width c) (height a)...

   _ <- submitTask (-) [a,b,tmp1]
   _ <- submitTask (+) [c,d,tmp2]
   t <- submitTask (*) [tmp1,tmp2,r]

   waitTask t -- or we could use a callback
   releaseData [tmp1,tmp2]
```

Motivating example

Enhanced data management

Suppose we want to compute (a, b, c and d are matrices):

```
r = (a-b) * (c+d)
```

Instead of (pseudo-code):

```
do
   tmp1 <- allocateMatrixData (width a) (height a)...
   tmp2 <- allocateMatrixData (width c) (height c)...
   r <- allocateMatrixData (width c) (height a)...

   _ <- submitTask (-) [a,b,tmp1]
   _ <- submitTask (+) [c,d,tmp2]
   t <- submitTask (*) [tmp1,tmp2,r]

   waitTask t -- or we could use a callback
   releaseData [tmp1,tmp2]
```

We would write:

```
r = (a-b) * (c+d)
```

Motivating example

Data dependent control

Suppose we want to compute (a,b and c are matrices):

```
r = if (f a) then (add b c) else (sub b c)
```

Motivating example

Data dependent control

Suppose we want to compute (a,b and c are matrices):

```
r = if (f a) then (add b c) else (sub b c)
```

Instead of (pseudo-code):

```
do
   tmp <- allocateBoolData
   t1 <- submitTask f [a,tmp]
   waitTask t1 -- or we could use a callback
   tmpHost <- retrieveBoolData tmp

   r <- allocateMatrixData (width b) (height b)...
   if tmpHost
      then submitTask add [b,c,r]
      else submitTask sub [b,c,r]

   releaseData [tmpHost,tmp]
```

Motivating example

Data dependent control

Suppose we want to compute (a,b and c are matrices):

```
r = if (f a) then (add b c) else (sub b c)
```

Instead of (pseudo-code):

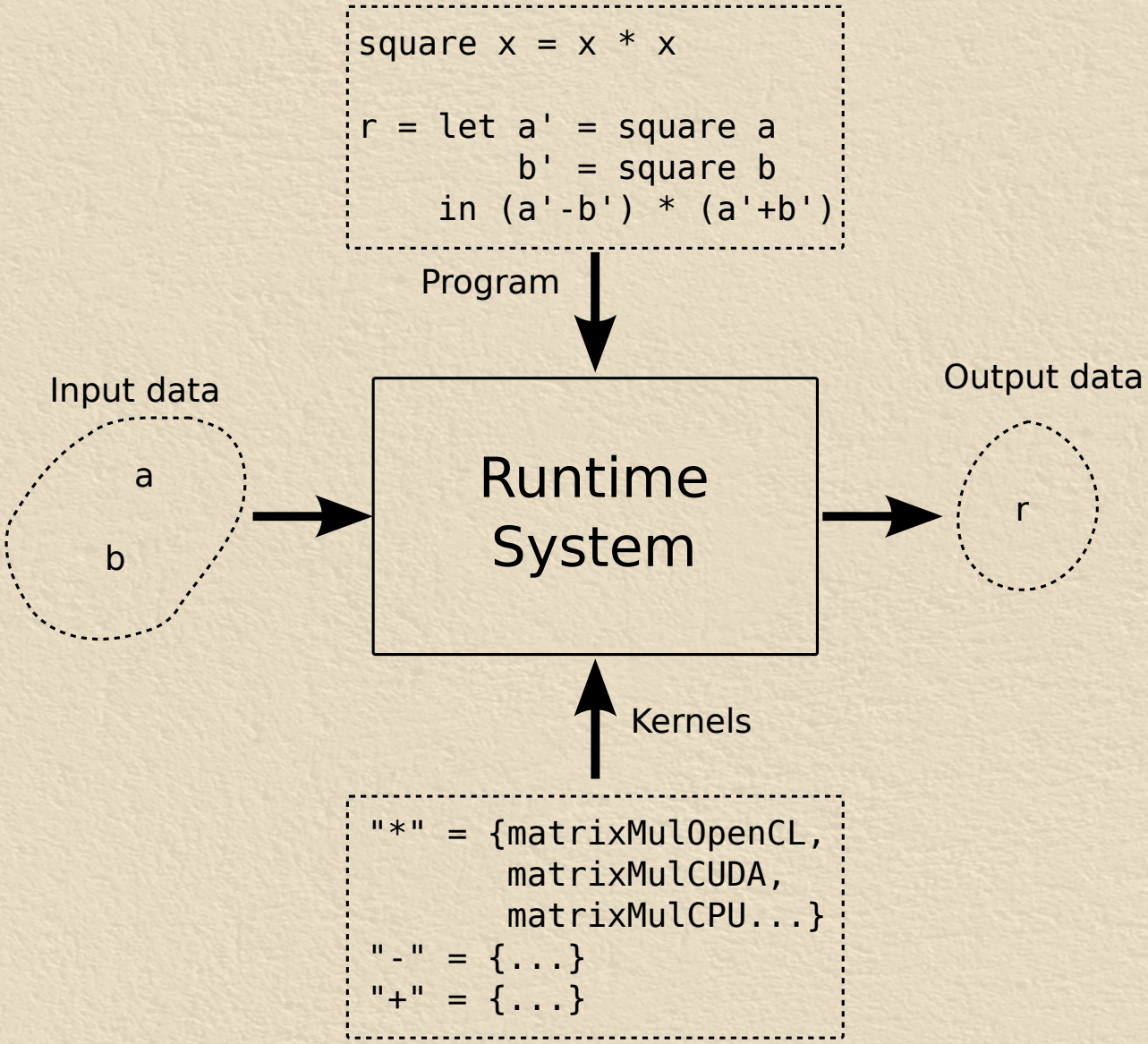
```
do
   tmp <- allocateBoolData
   t1 <- submitTask f [a,tmp]
   waitTask t1 -- or we could use a callback
   tmpHost <- retrieveBoolData tmp

   r <- allocateMatrixData (width b) (height b)...
   if tmpHost
      then submitTask add [b,c,r]
      else submitTask sub [b,c,r]

   releaseData [tmpHost,tmp]
```

We would write:

```
r = if (f a) then (add b c) else (sub b c)
```

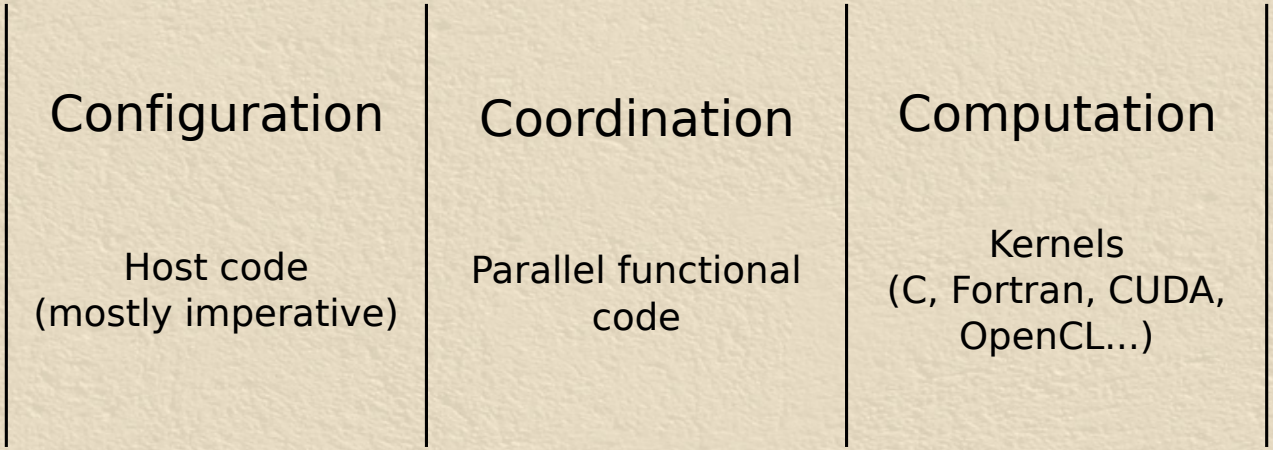
ViperVM

Overview



ViperVM

Overview



ViperVM

Host code example

```
pf <- initPlatform $ Configuration {libraryOpenCL="libOpenCL.so"}
rt <- initRuntime pf eagerScheduler

a <- initFloatMatrix rt [[1.0, 2.0, 3.0],
                         [4.0, 5.0, 6.0],
                         [7.0, 8.0, 9.0]]

b <- initFloatMatrix rt [[1.0, 4.0, 7.0],
                         [2.0, 5.0, 8.0],
                         [3.0, 6.0, 9.0]]

builtins <- loadBuiltins rt [
    ("+", floatMatrixAddBuiltin),
    ("-", floatMatrixSubBuiltin),
    ("*", floatMatrixMulBuiltin),
    ("a", dataBuiltin a),
    ("b", dataBuiltin b)]

r <- eval builtins =<< readFile "codelet.vvm"

printFloatMatrix rt r
```

-- File: codelet.vvm

```
square x = x * x

main = let a' = square a
           b' = square b
       in (a'-b')*(a'+b')
```

ViperVM

Challenges

Challenge 1: avoid opaque host code

- ➔ Host code limited to runtime configuration
- ➔ Runtime has full control of parallel reduction

ViperVM

Challenges

Challenge 1: avoid opaque host code

- ➔ Host code limited to runtime configuration
- ➔ Runtime has full control of parallel reduction

Challenge 2: enhanced data management

- ➔ Output data automatically allocated
- ➔ Garbage collector

ViperVM

Challenges

Challenge 1: avoid opaque host code

- ➔ Host code limited to runtime configuration
- ➔ Runtime has full control of parallel reduction

Challenge 2: enhanced data management

- ➔ Output data automatically allocated
- ➔ Garbage collector

Challenge 3: automatic granularity adaptation

- ➔ Next slide

ViperVM

Automatic granularity adaptation

Idea

Substitute a redex (involving a kernel) with another expression

Example

Substitute `a+b` with:

```
unsplit (add' a' b')
  where
    a' = split w h a
    b' = split w h b
    add' = zipWith (zipWith (+))
```

Example: Tiled Matrix Addition (tile size = 8k)

```
/* StarPU */
struct starpu_data_filter f = {
    .filter_func = starpu_matrix_filter_vertical_block,
    .nchildren = nw // w div 8192
};

struct starpu_data_filter f2 = {
    .filter_func = starpu_matrix_filter_block,
    .nchildren = nh // h div 8192
};

starpu_data_map_filters(a, 2, &f, &f2);
starpu_data_map_filters(b, 2, &f, &f2);
starpu_data_map_filters(c, 2, &f, &f2);

for (i=0; i<nw; i++) { for (j=0; j<nh; j++) {
    starpu_data_handle_t sa = starpu_data_get_sub_data(a, 2, i, j);
    starpu_data_handle_t sb = starpu_data_get_sub_data(b, 2, i, j);
    starpu_data_handle_t sc = starpu_data_get_sub_data(c, 2, i, j);

    starpu_insert_task(&add, STARPU_R, sa, STARPU_R, sb, STARPU_W, sc, 0);
}}

starpu_task_wait_for_all();
starpu_data_unpartition(c,0);
```

```
-- ViperVM: with naive automatic
-- granularity adaptation
c = a + b
```

```
-- ViperVM: explicit
c = unsplit (add' a' b')
    where a' = split 8192 8192 a
          b' = split 8192 8192 b
          add' = zipWith (zipWith (+))
```

Dimensions	ViperVM 3 GPUs+CPU	ViperVM 3 GPUs	StarPU 3 GPUs
16K x 16K	1.9s	2.1s	1.4s
24K x 24K	4.0s	4.4s	2.9s

ViperVM

Status

- Alpha version 0.2
  - `https://github.com/hsyl20/HViperVM/tree/0.2`
- Implemented
  - Parallel reducer (using STM)
  - OpenCL backend
  - Eager scheduling strategy
  - Lisp-like frontend (parser)
  - Basic single-precision linear algebra OpenCL kernels
  - Naive substitution mechanism (based on input sizes)
- Missing
  - Garbage collector
  - Other backends (CUDA, Xeon Phi...)
  - Better scheduling strategies (HEFT...)
  - Other frontends (with type checking, etc.)
  - ...

ViperVM

Future work (long term)

- Integration with automatic kernel generation
  - e.g. Data.Array.Accelerate
  - $T^f = \{K^f_{x86}, K^f_{OpenCL}, K^f_{CUDA}, K^f_{Accelerate} \cdots\}$
- Automatic check-pointing
  - Save current reduction state + required data to restart

Thank you

Questions?