
SIGGRAPH2015
Xroads of Discovery

SIGGRAPH2015
Xroads of Discovery

The 42nd International Conference and Exhibition
on Computer Graphics and Interactive Techniques

Presto Execution System

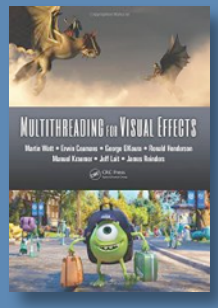
An Asynchronous Computation Engine for Animation

George ElKoura
Pixar

More Info

@multithreadvfx

www.multithreadingandvfx.org



“Multithreading for Visual Effects”

Team

Engineering

Florian Zitzelsberger Florian Sauer John Loy

Thanks

Steve May Guido Quaroni Adam Woodbury
Dirk Van Gelder Alicia Mooty

Pixar Software R&D

Overview

- What is Presto?
- Execution system architecture
- Multithreading strategies
- Switching a large code base to TBB

Presto Execution

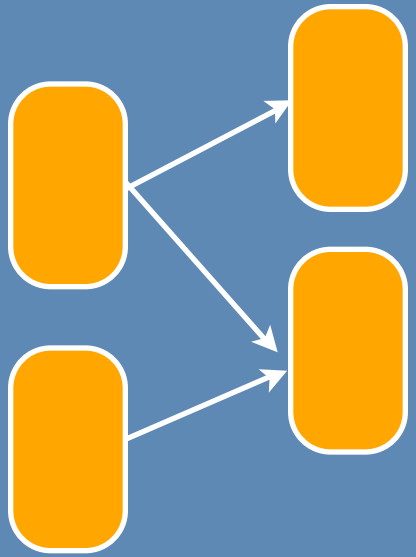
- The Presto Execution System:
  - A general purpose execution engine
  - Commonly used for posing points
  - Must be as fast as possible

Phases of Execution

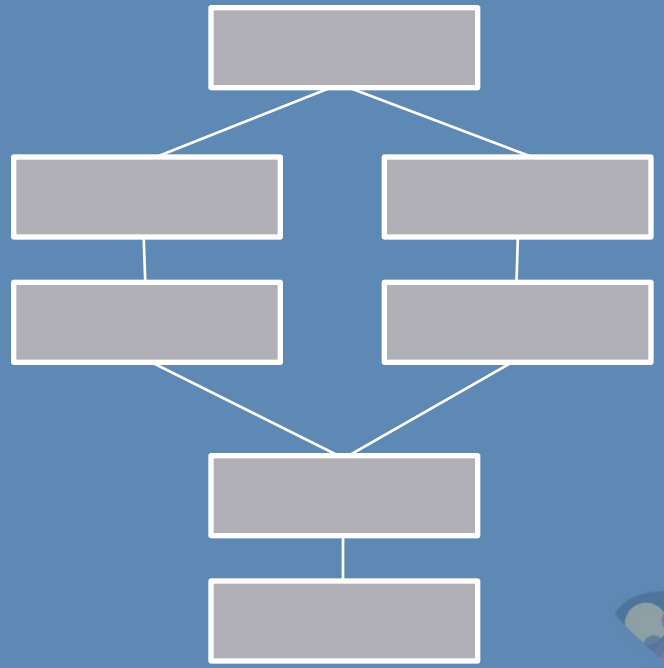
- Execution is broken up into three phases:
  - Compilation
  - Scheduling
  - Evaluation

Compilation

User-Authored Scene Objects



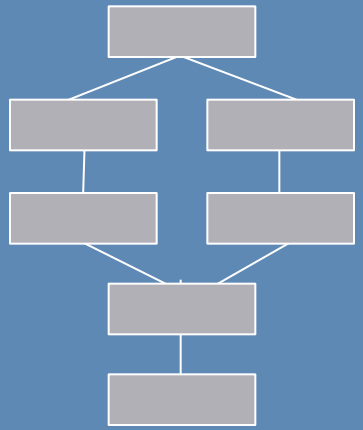
Compilation



Compiled Data Flow Network

Scheduling

Compiled Network

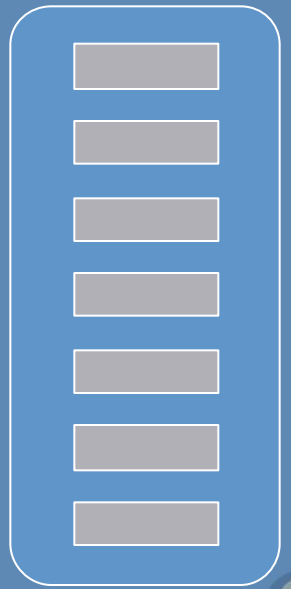


+

Client Request

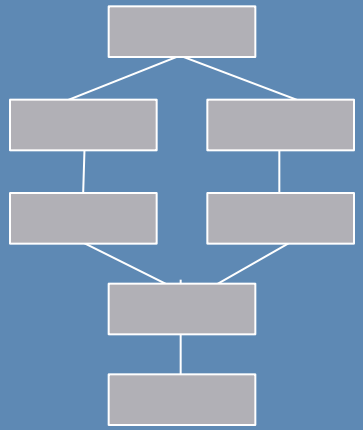
Request
Body points
Body transform
Head points
Head transform

Scheduling



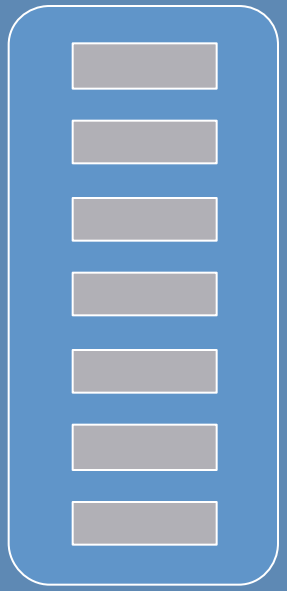
Evaluation

Compiled Network



+

Schedule



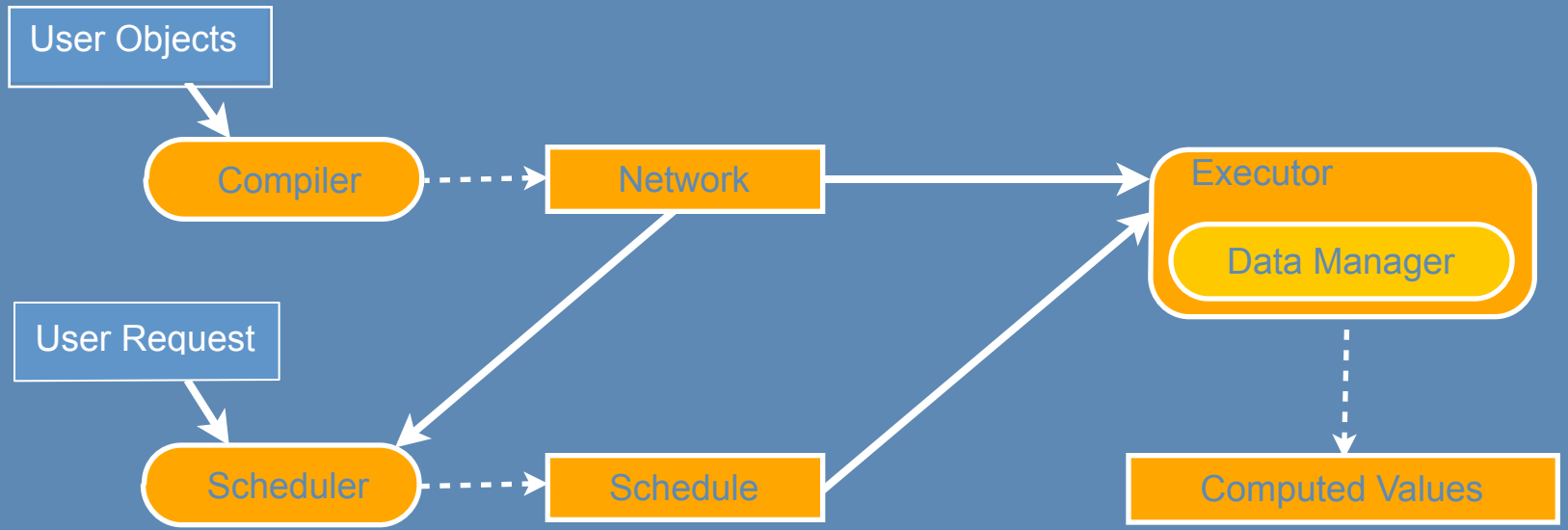
Evaluation

Results

Computed Values

(1.4, 0.0, 2.4)
(1.4, -0.784, 2.4)
(0.784, -1.4, 2.4)
(0.0, -1.4, 2.4)
(1.3375, 0.0, 2.53125)
(1.3375, -0.749, 2.53125)
(0.749, -1.3375, 2.53125)
(1.4375, 0.0, 2.53125)
(1.4375, -0.805, 2.53125)
(0.805, -1.4375, 2.53125)
(0.0, -1.4375, 2.53125)
(1.5, 0.0, 2.4)
...

Exec Engine Architecture



User Extensions

- Plugin support
- Need painless multithreading
- Provide safeguards to help

Execution Callbacks

```
static void
MyCallback(const Context &context)
{
    ...
}
```

Execution Callbacks

- Iterators are provided for vectorized data access...
- which allows for easier experimentation with multithreading strategies.

User Gotchas

- Avoid global and static variables
- Be careful about singletons
- Using a non-thread-safe 3rd party library requires users to do their own locking
- ... and, of course, Python

Python in Execution

- Global Interpreter Lock (GIL)
- Only one thread at a time
- Do heavy lifting in C++
- Look out for GIL deadlocks

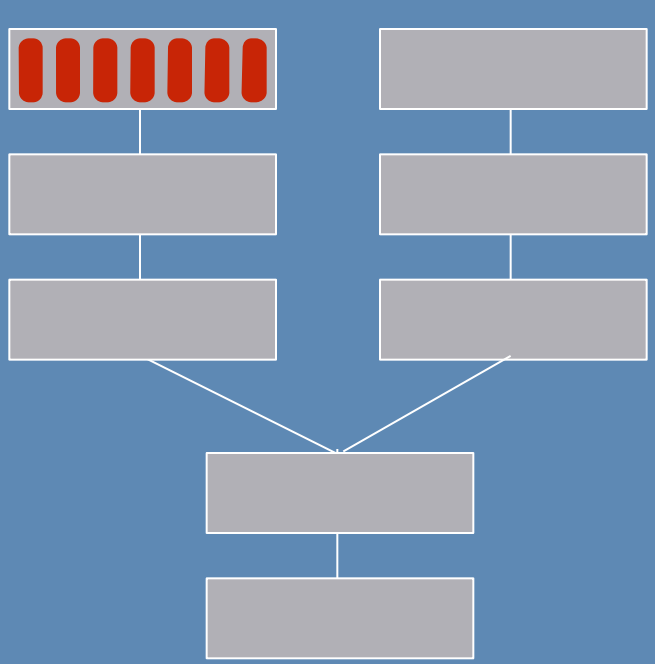
Flexibility to Experiment

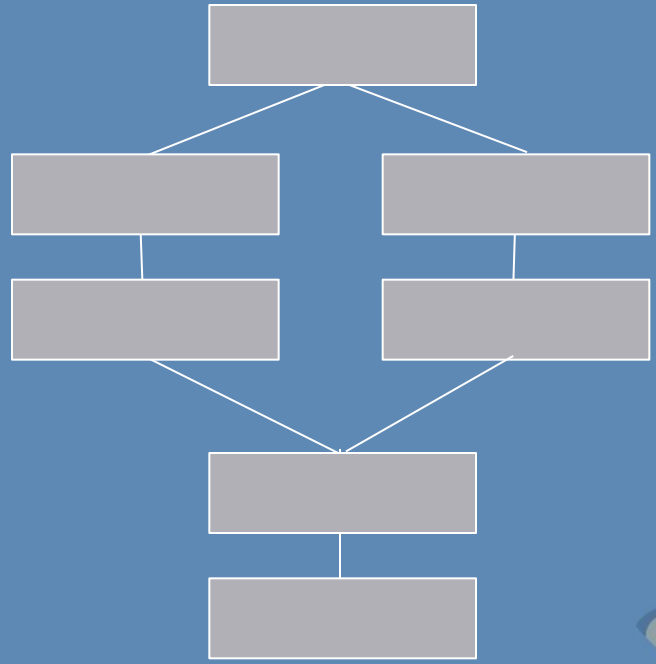
- Modular, adaptable design, e.g.:
  - Hardware changes: rewrite executor, possibly scheduler
  - User representation changes: rewrite compilation
- Allows for easy experimentation with different algorithms

Multithreading

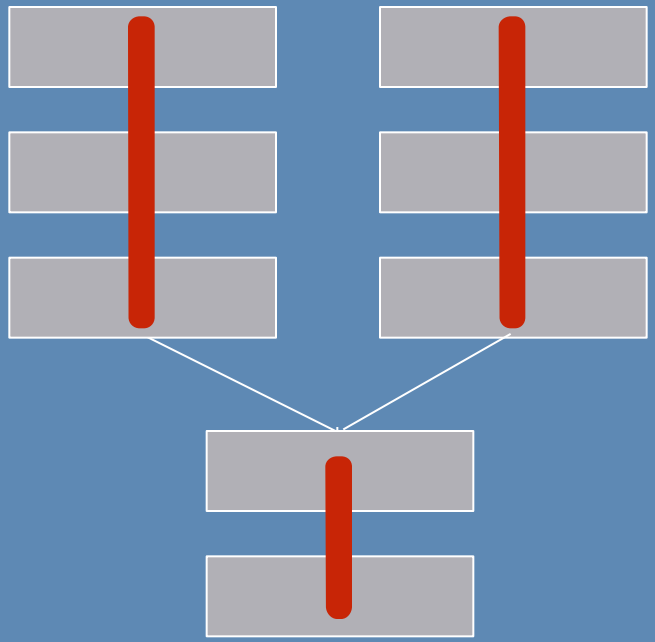
- Computed data not in the network
- Stateless computations
- Modular schedulers and executors

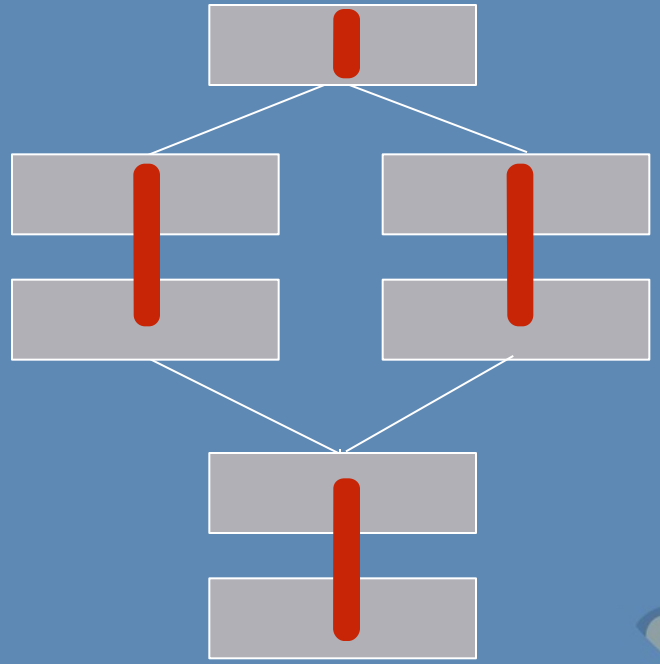
Per Node



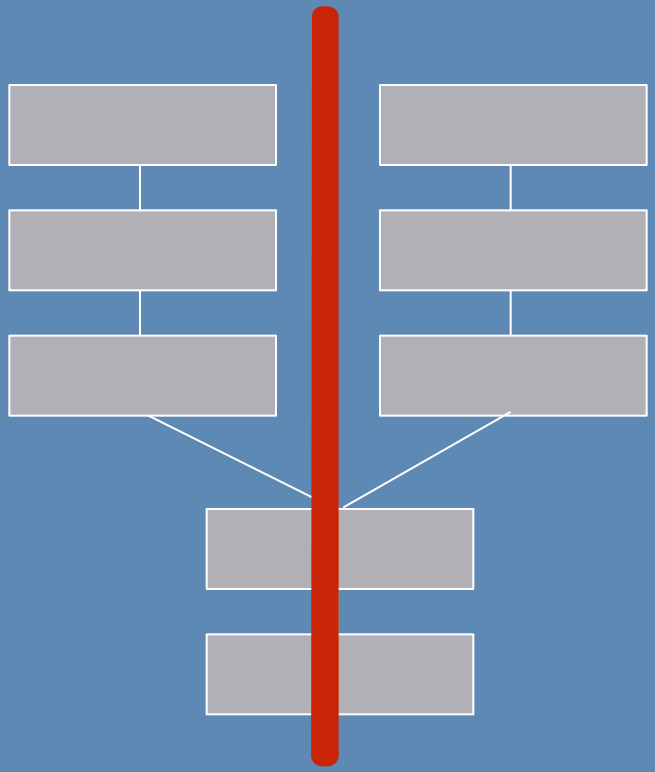


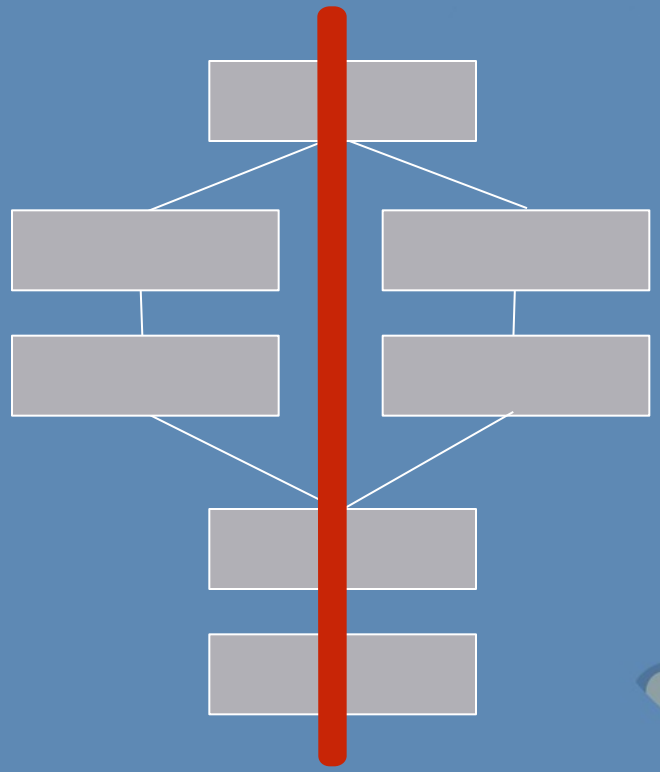
Per Branch



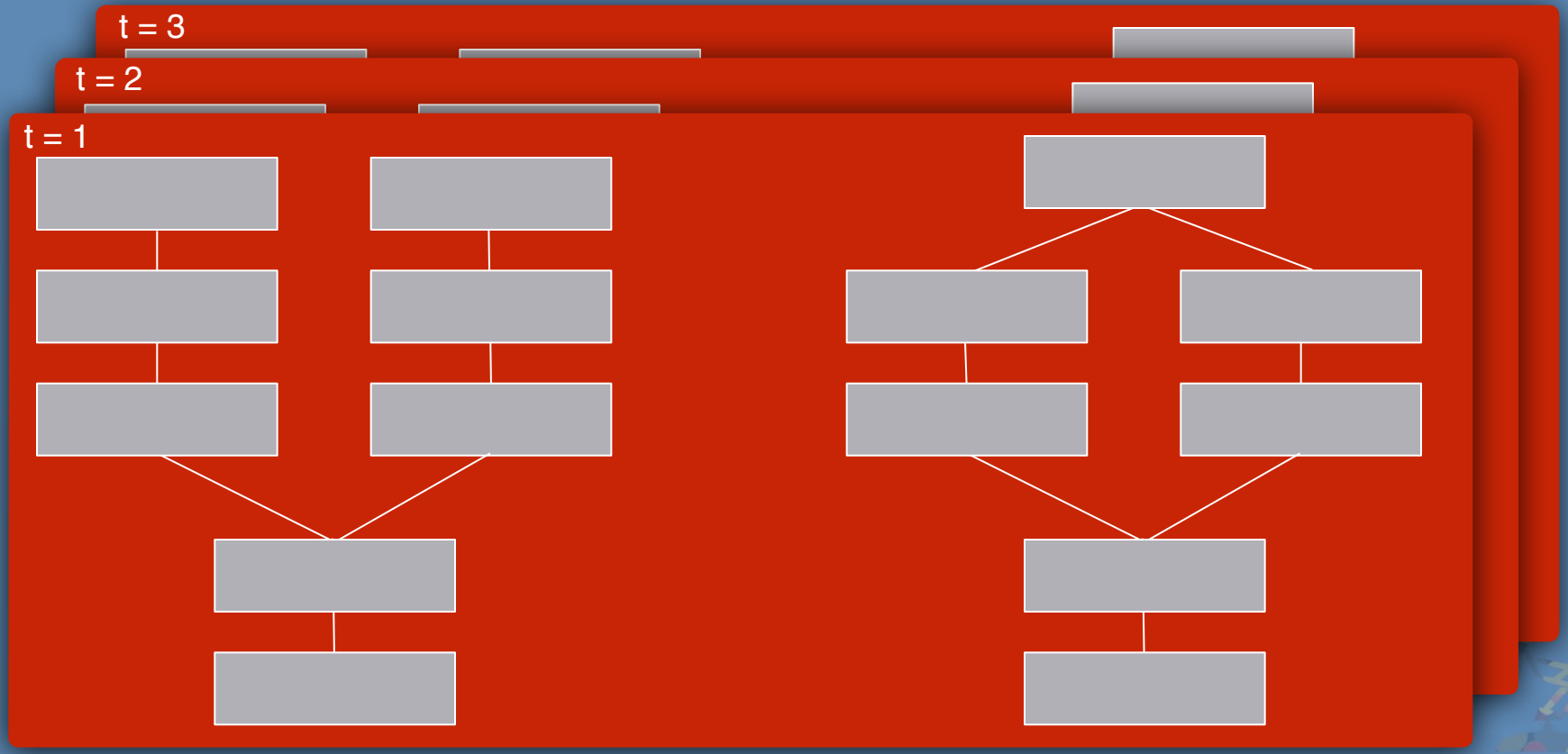


Per Model





Per Frame



Background Execution

- Evaluates entire frames at different times in parallel and in the background
- Fills in a computed value cache

Interruption

- If values change, we must interrupt, but…
- We can’t block to wait for it to interrupt.

Interruption

- Avoid `pthread_cancel` and `pthread_kill`
- Instead, we use a cooperative scheme...
- and throw away results.

TBB

Switching to TBB

- We had our own threading toolkit
- Built on for a decade or two
- Lacked modern features
- Wanted to switch to TBB

libWork

1. Create abstraction
2. Convert clients
   - Don’t disrupt shipping software
3. Implement TBB backend
4. Flip the switch

libWork Abstractions

- Parallel For
- Fixed Dispatcher
- Dynamic Dispatcher
- Background Job

Switching to TBB is Easy!

```
#define WORK_USE_TBB ~~0~~ 1
```

Unexpected Deadlocks

```
void F()
{
    // do something ...
    A();
    // do more things ...
}
```

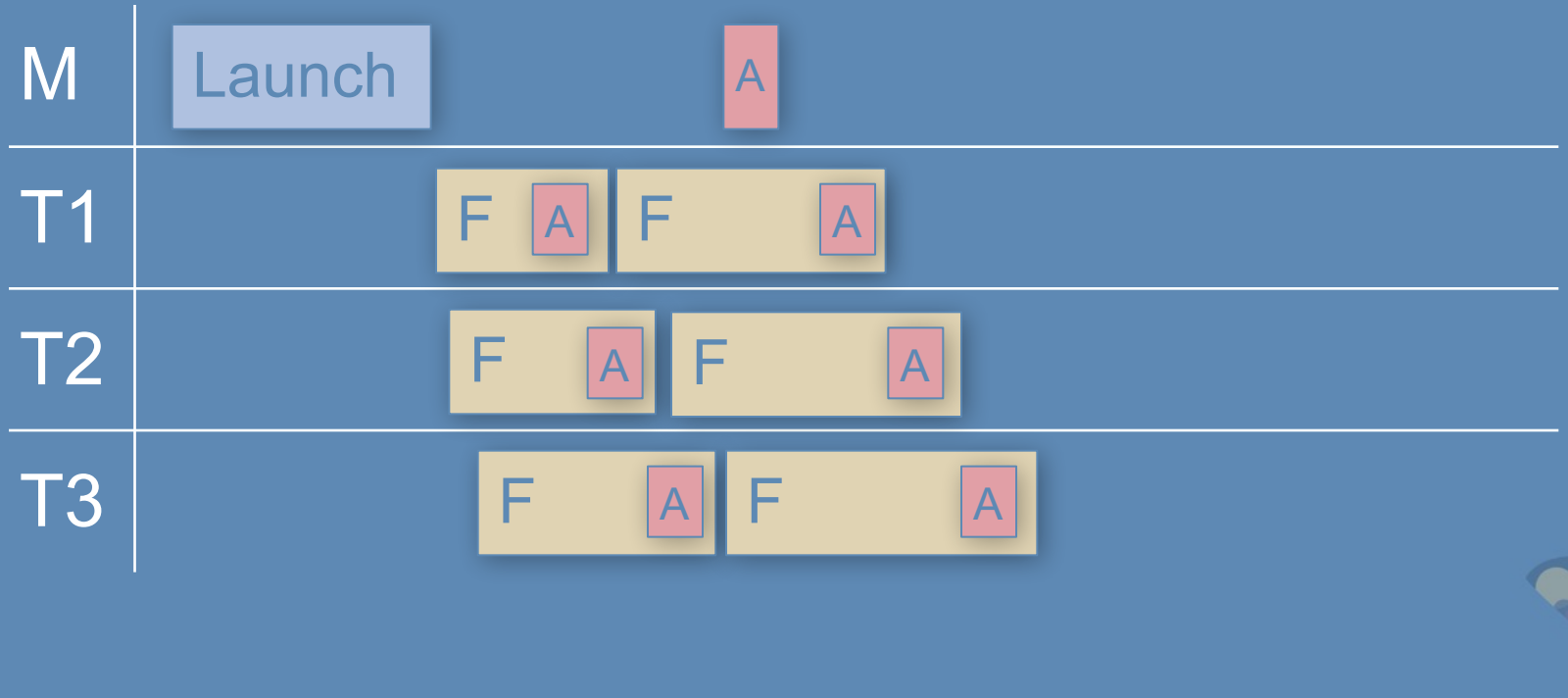
Unexpected Deadlocks

```
A () {
    Lock a;
    if (cache.Lookup())
        return hit;
    parallel_for([]{ fill in cache })
    return hit;
}
```

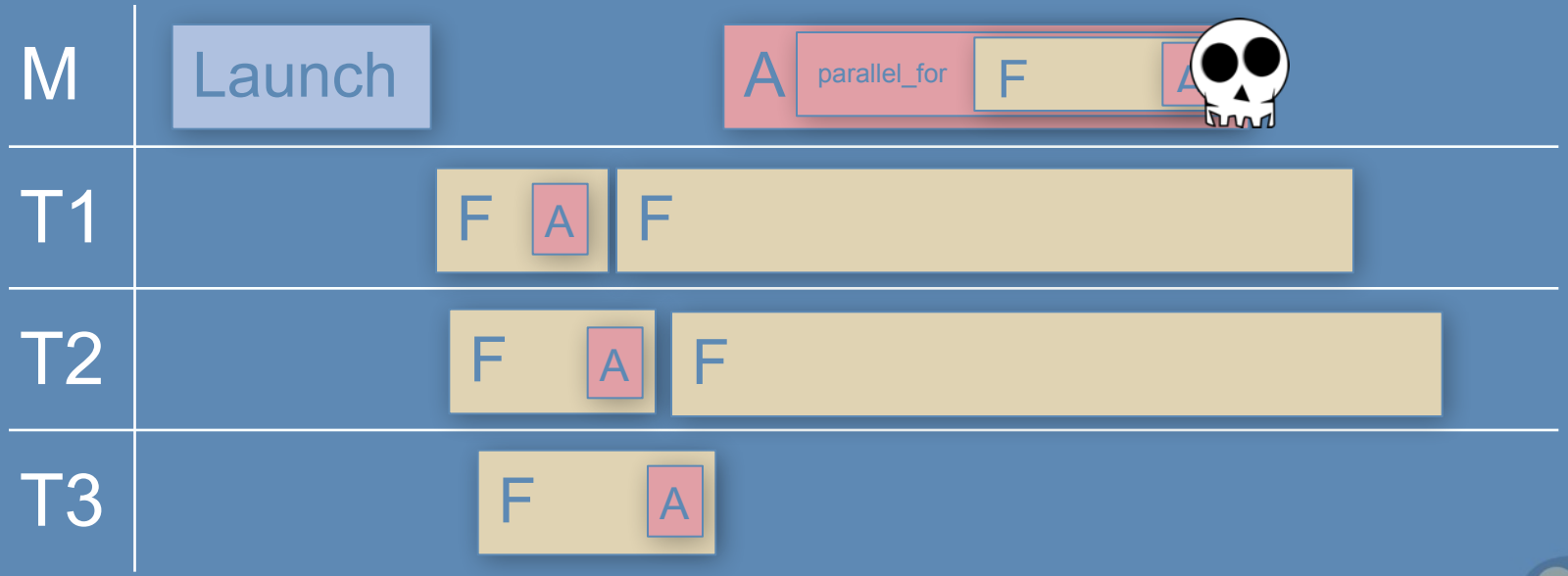
Unexpected Deadlocks

```
int main() {
    …
    Spawn_many_tasks_running_F();
    A();
    Wait_for_Fs();
    …
}
```

Without Work Stealing



With Work Stealing



Unexpected Deadlocks

```
A () {
    Lock a;
    if (cache.Lookup())
        return hit;
    parallel_for([]{ fill in cache })
    return hit;
}
```

not smart!

Thread Throttling

- Had strict control
- TBB hungry
- tbb::task_scheduler_init()

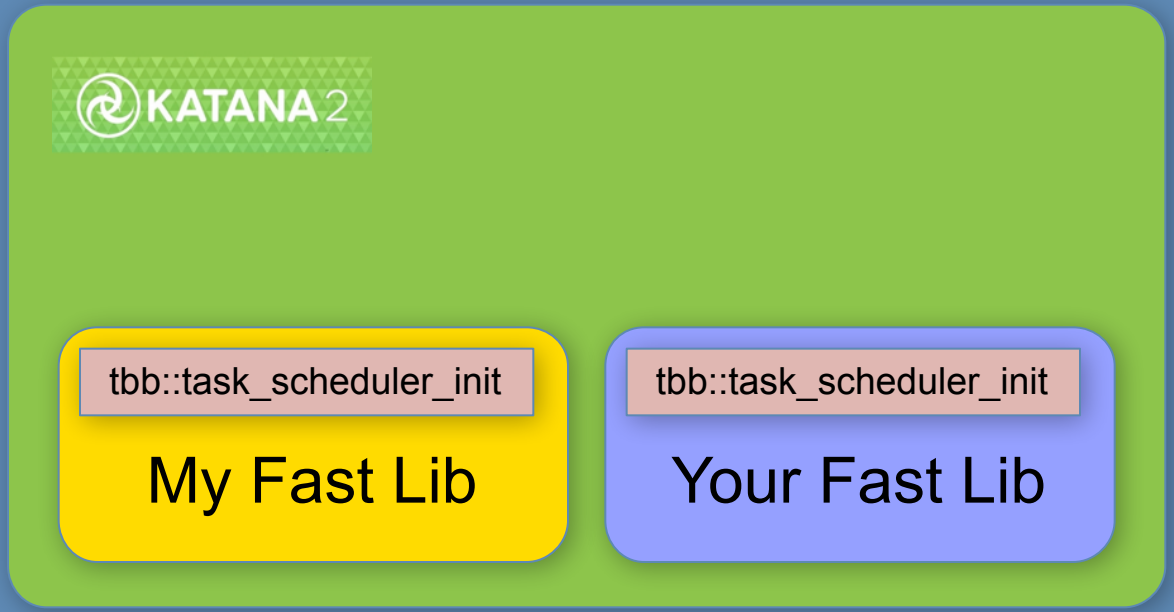
tbb::task_scheduler_init

- Hard in embedded libs
- Stack object, must go first
- Who goes first?

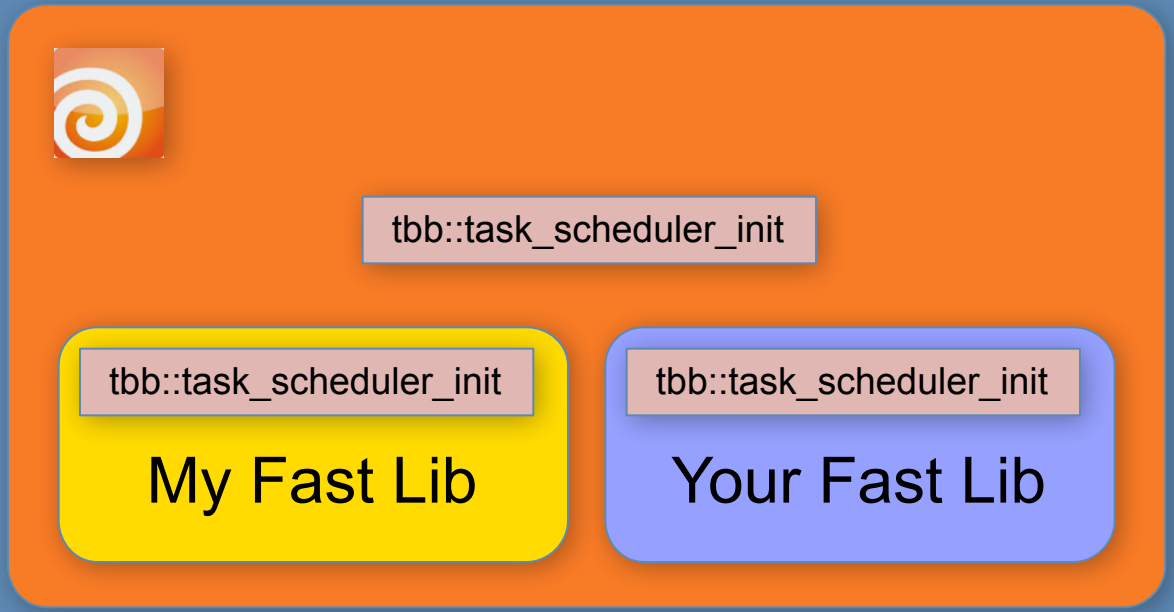
In Practice

In Practice

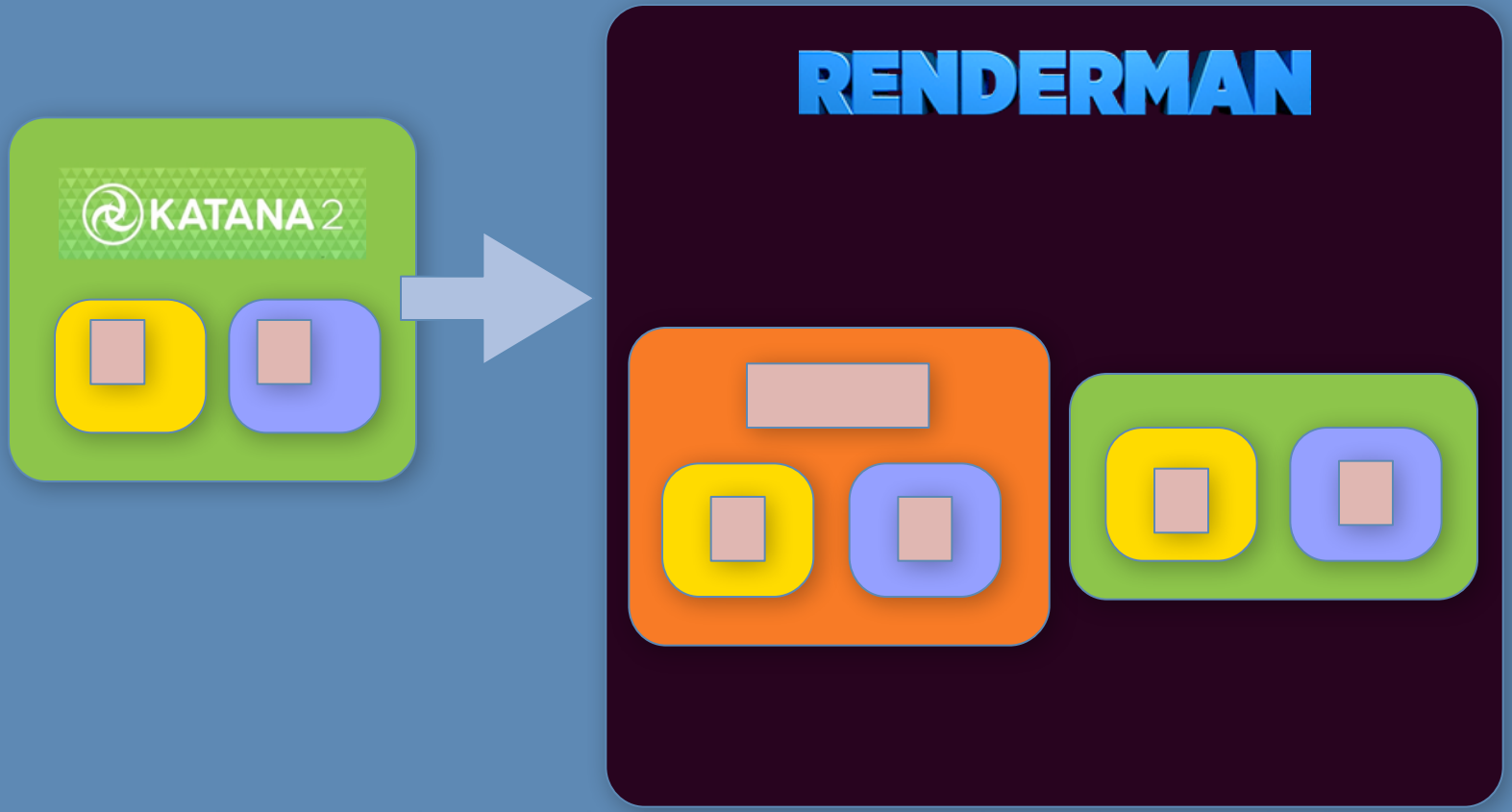
In Practice


KATANA 2
tbb::task_scheduler_init
My Fast Lib
tbb::task_scheduler_init
Your Fast Lib

In Practice


tbb::task_scheduler_init
tbb::task_scheduler_init
My Fast Lib
tbb::task_scheduler_init
Your Fast Lib

All on one machine...



What we’re doing

- Don’t oversubscribe the farm
- Centralize decision making
- So far okay, not great

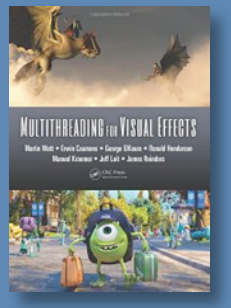
Recap

- Separate your data from your algorithm
- Build modular systems
- TBB is great
- Think in tasks, not threads
- Careful with resource management

THANKS!

@multithreadvfx

www.multithreadingandvfx.org



“Multithreading for Visual Effects”