AMD
GPUOpen

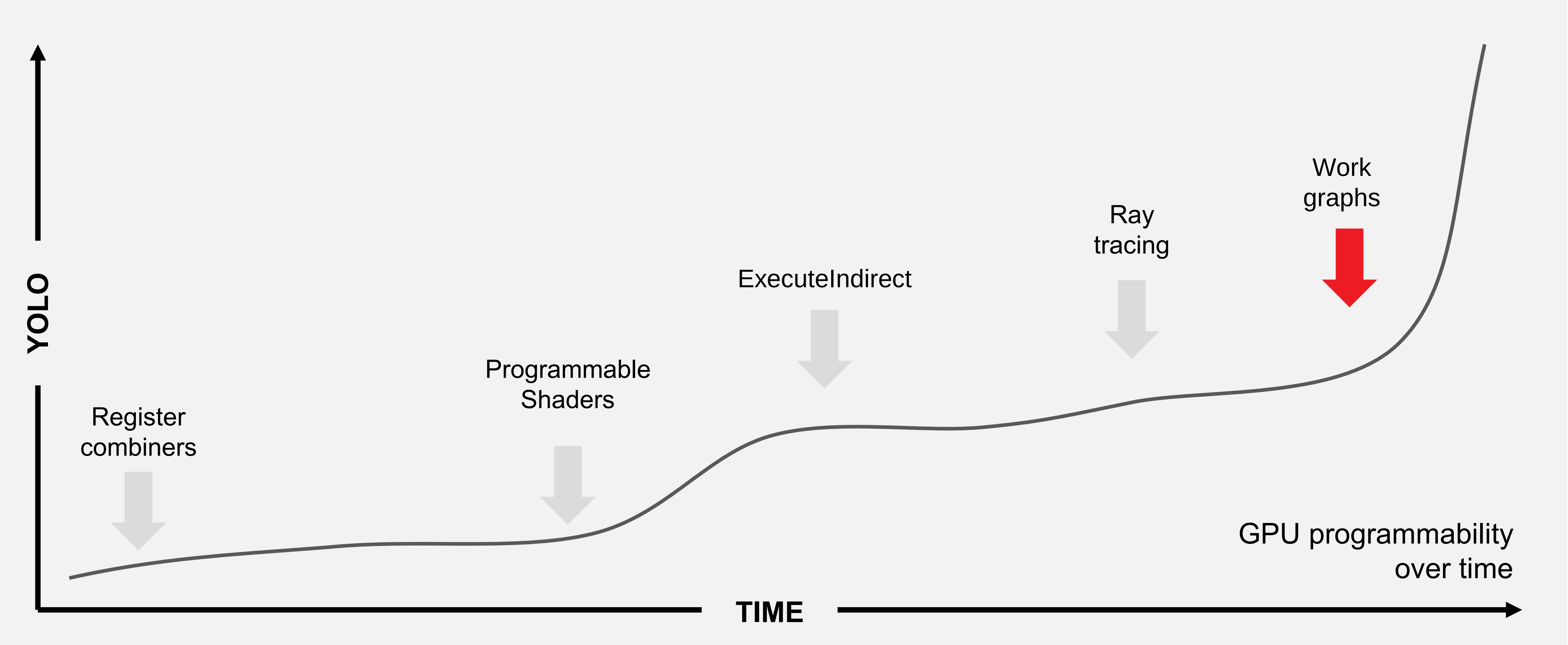
GDC MARCH 18-22, 2024
SAN FRANCISCO, CA

GPU Work Graphs: Welcome to the Future of GPU Programming

Matthäus G. Chajdas, AMD
Shawn Hargreaves, Microsoft

#GDC2024

A NEW DAWN!


YOLO
TIME
Register combiners
Programmable Shaders
ExecuteIndirect
Ray tracing
Work graphs
GPU programmability over time

WORK GRAPH MOTIVATION

> “If only I could launch work on the GPU”

— Most game developers over the last few years ☺

WORK GRAPH MOTIVATION

> "I can launch GPU work using ExecuteIndirect!"

> "Wow, this is an awful programming model…"

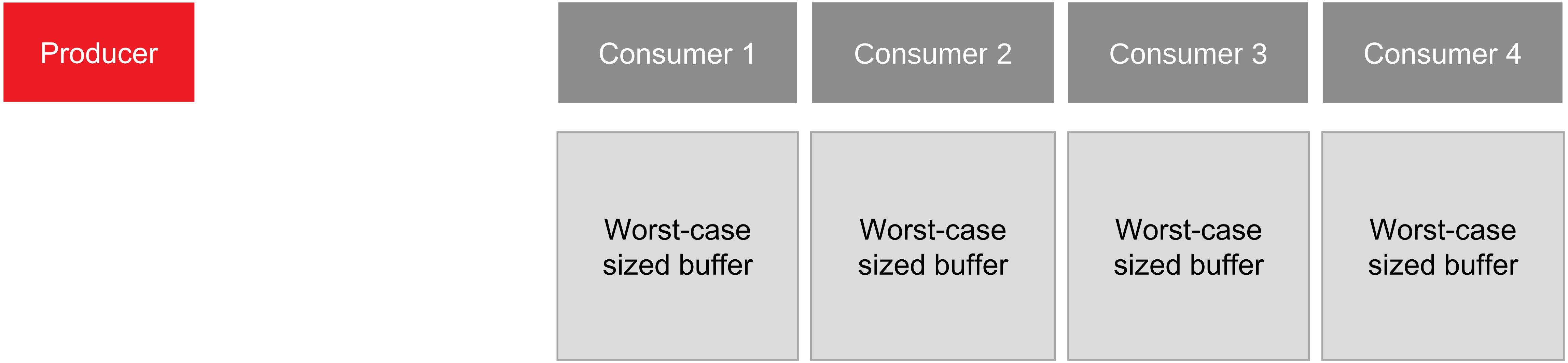
— Experienced game developers

WORK GRAPH MOTIVATION

> “ExecuteIndirect is an awful programming model.”
>
> — Hardware designers
>
> — Driver developers
>
> — Authors of every GPU debugging tool

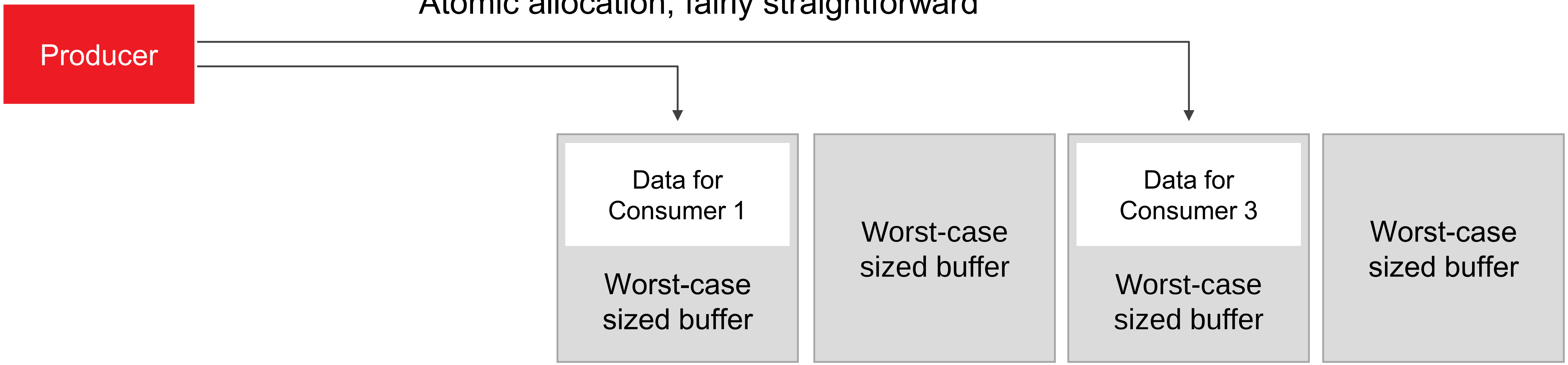
WHAT’S THE PROBLEM?

Classify work into one of the several buckets,
for example, based on shader complexity.



WHAT’S THE PROBLEM?

Producer writes data into consumer buffers
Atomic allocation, fairly straightforward



Producer

Data for Consumer 1

Worst-case sized buffer

Worst-case sized buffer

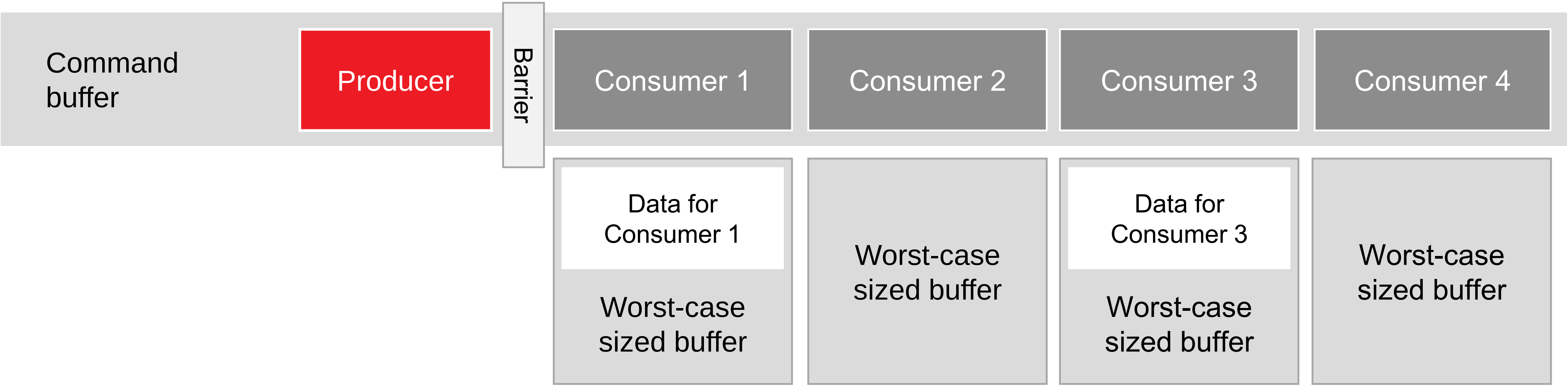
Data for Consumer 3

Worst-case sized buffer

Worst-case sized buffer

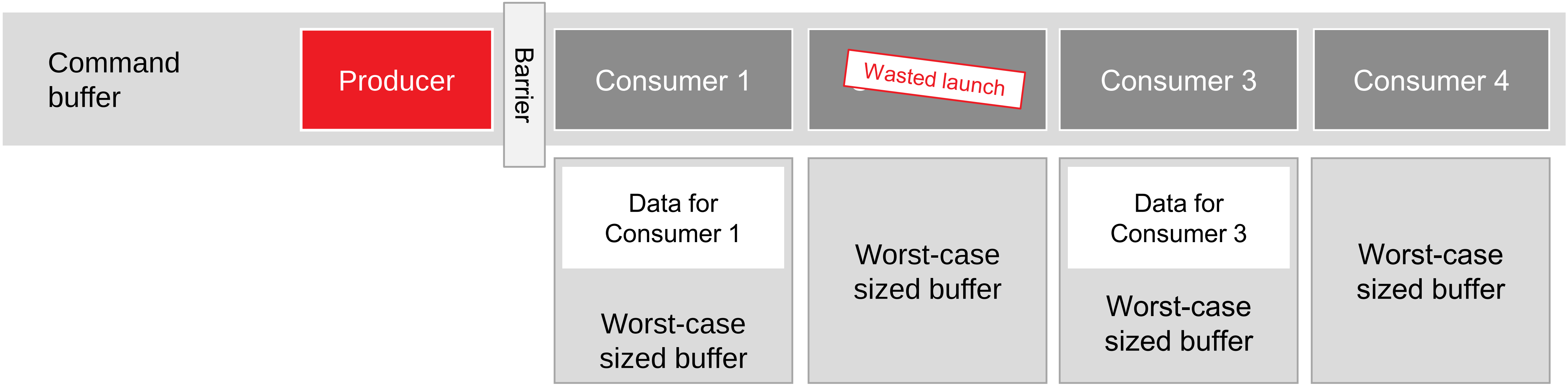
WHAT’S THE PROBLEM?

Barrier between producer and consumers,
empty launch overhead, wasted memory, lost locality…



WHAT’S THE PROBLEM?

Barrier between producer and consumers,
empty launch overhead, wasted memory, lost locality…



Command buffer

Producer

Barrier

Consumer 1

Wasted launch

Consumer 3

Consumer 4

Data for Consumer 1

Worst-case sized buffer

Worst-case sized buffer

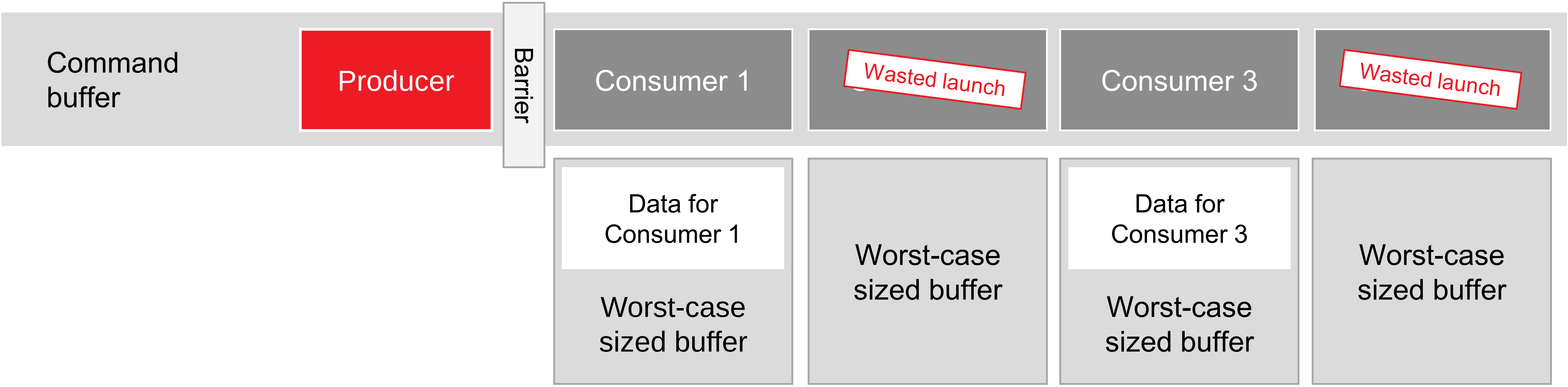
Data for Consumer 3

Worst-case sized buffer

Worst-case sized buffer

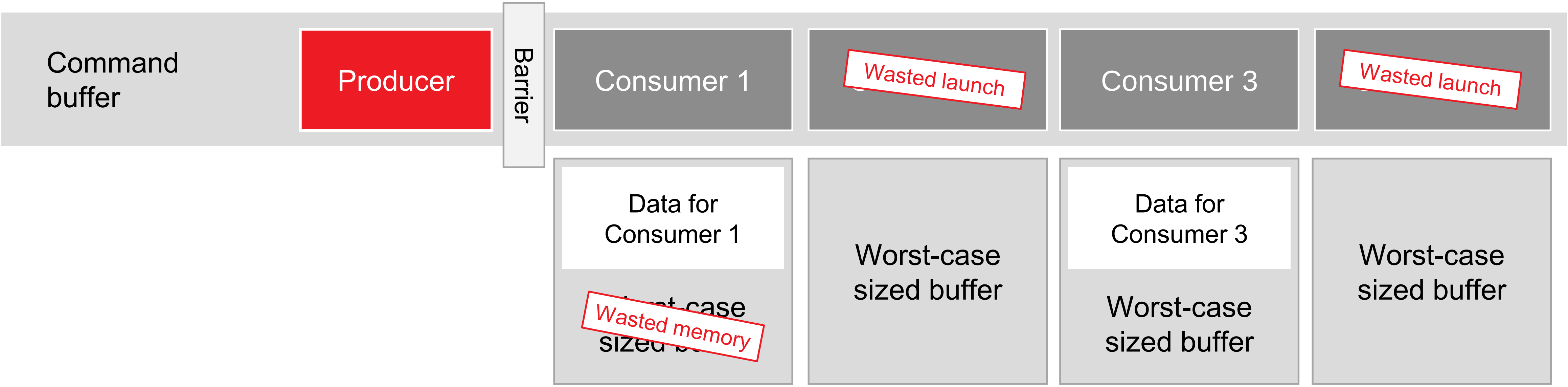
WHAT’S THE PROBLEM?

Barrier between producer and consumers,
empty launch overhead, wasted memory, lost locality…



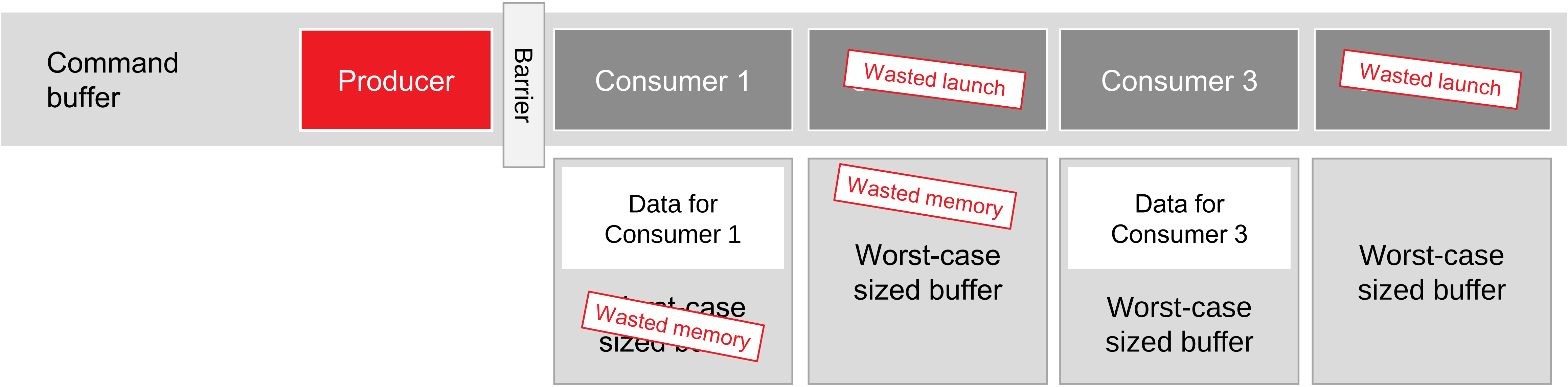
WHAT’S THE PROBLEM?

Barrier between producer and consumers,
empty launch overhead, wasted memory, lost locality…



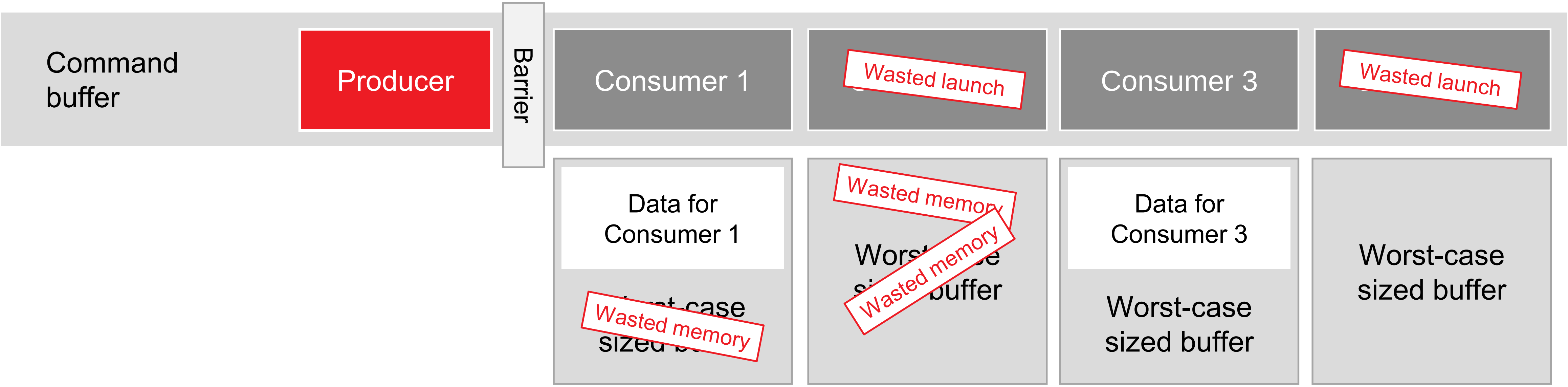
WHAT'S THE PROBLEM?

Barrier between producer and consumers, empty launch overhead, wasted memory, lost locality…



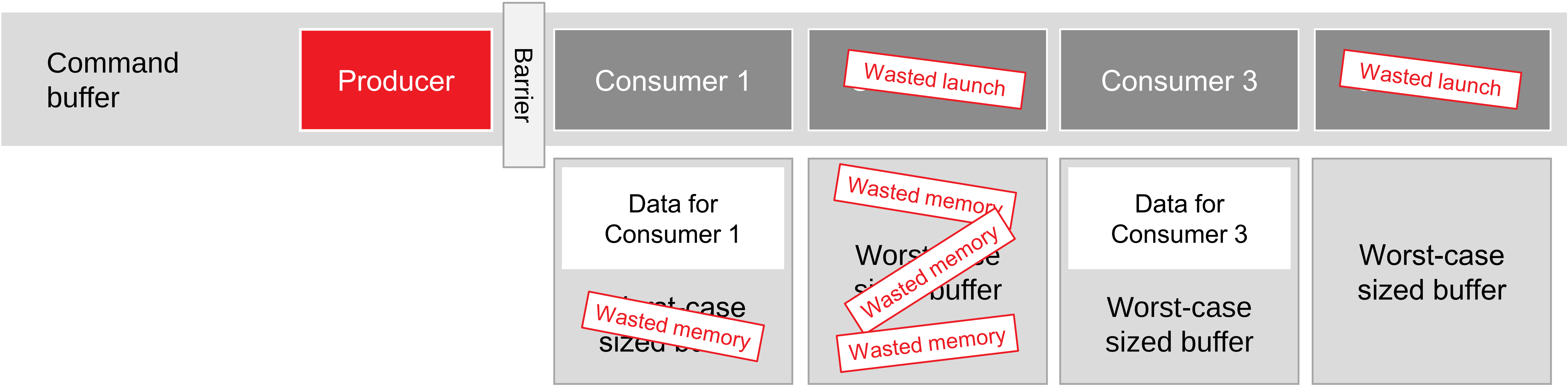
WHAT’S THE PROBLEM?

Barrier between producer and consumers, empty launch overhead, wasted memory, lost locality…



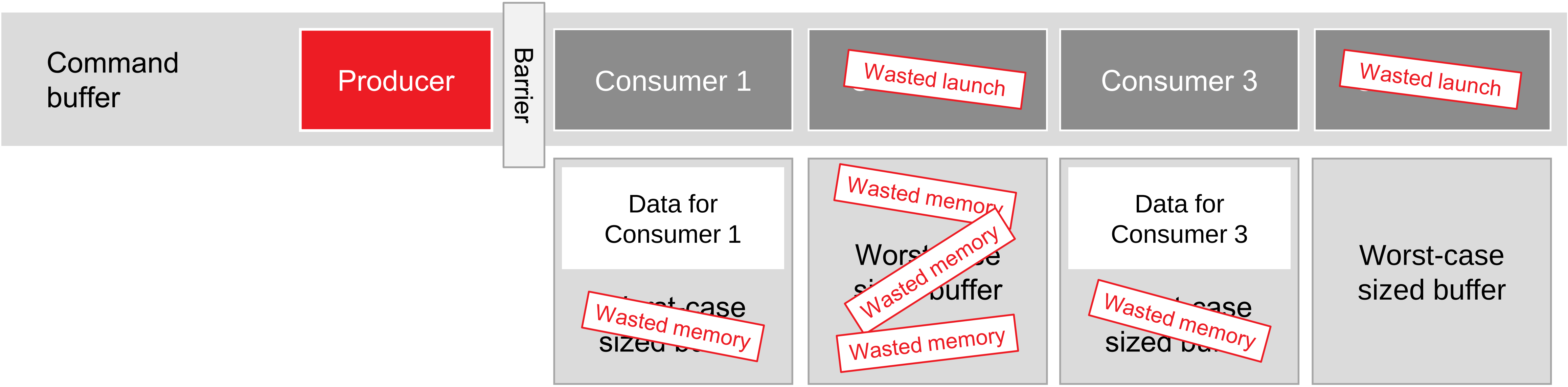
WHAT'S THE PROBLEM?

Barrier between producer and consumers,
empty launch overhead, wasted memory, lost locality…



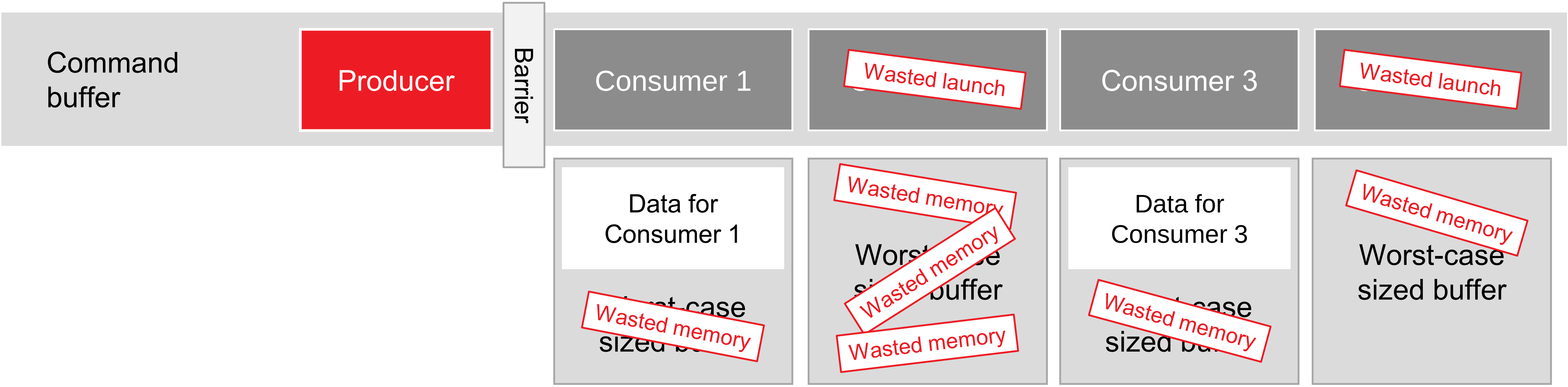
WHAT'S THE PROBLEM?

Barrier between producer and consumers, empty launch overhead, wasted memory, lost locality…



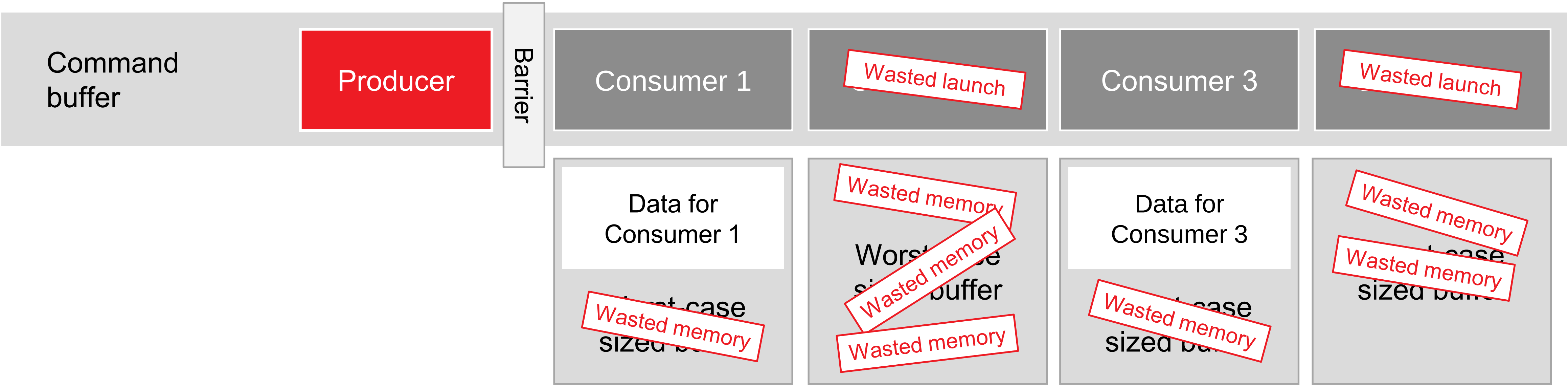
WHAT'S THE PROBLEM?

Barrier between producer and consumers,
empty launch overhead, wasted memory, lost locality…



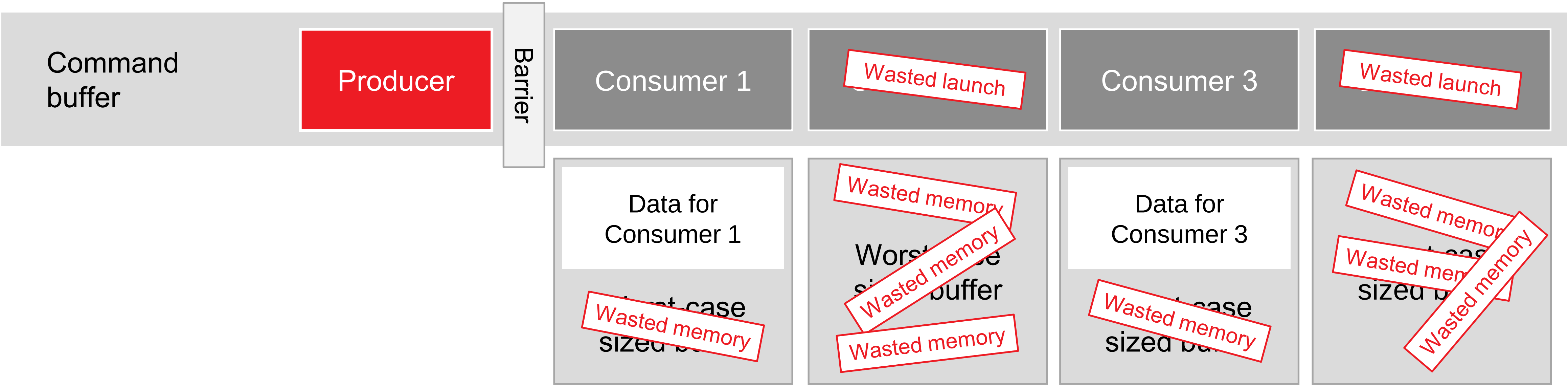
WHAT'S THE PROBLEM?

Barrier between producer and consumers,
empty launch overhead, wasted memory, lost locality…



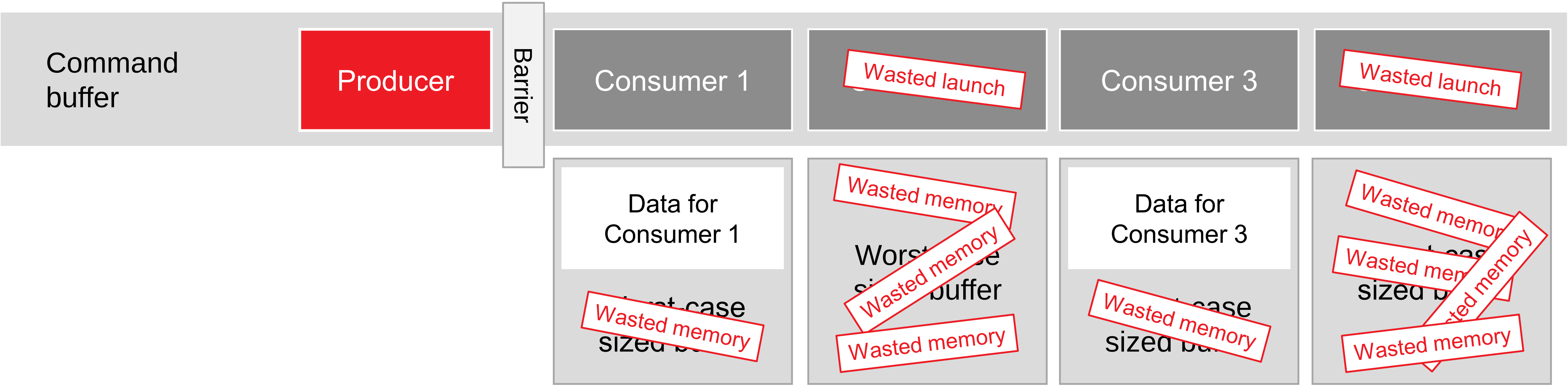
WHAT’S THE PROBLEM?

Barrier between producer and consumers, empty launch overhead, wasted memory, lost locality…



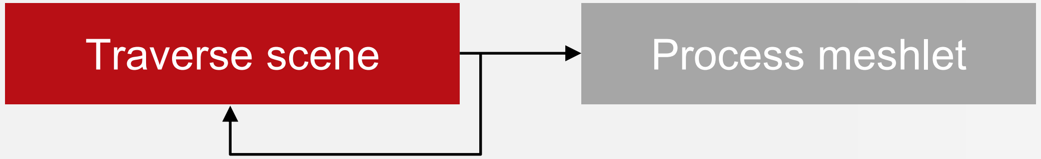
WHAT’S THE PROBLEM?

Barrier between producer and consumers,
empty launch overhead, wasted memory, lost locality…



~~PROBLEMS?~~ OPPORTUNITIES!

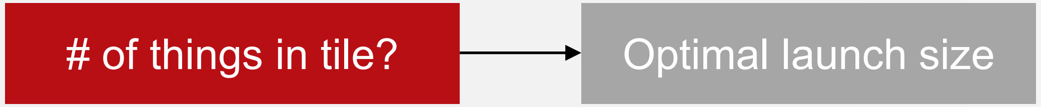
Recursive algorithms: Scene traversal, …



Traverse scene

Process meshlet

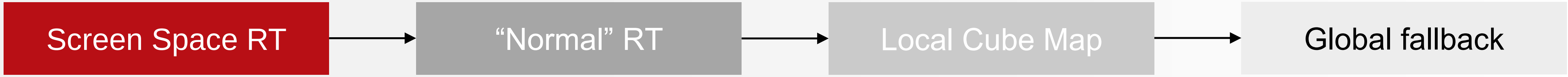
Adaptive algorithms (launch more/less work): Physics, …



# of things in tile?

Optimal launch size

Long execution chains: Lighting algorithms, …



Screen Space RT

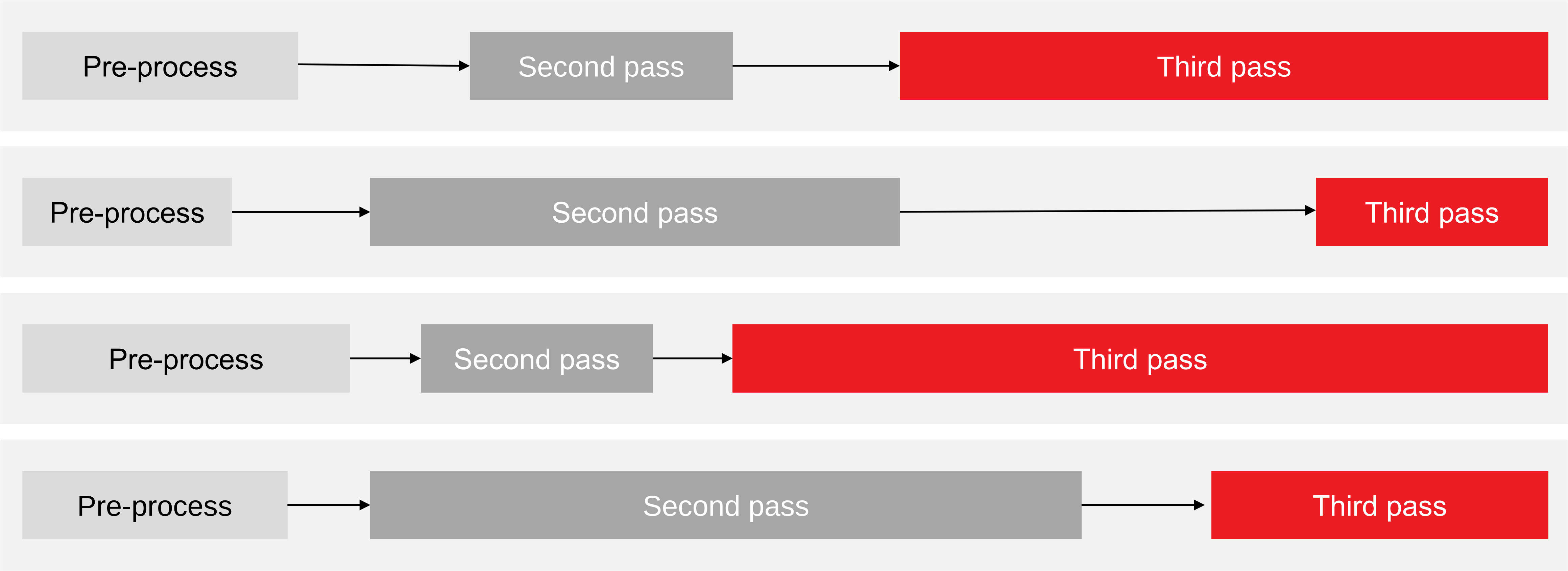
“Normal” RT

Local Cube Map

Global fallback

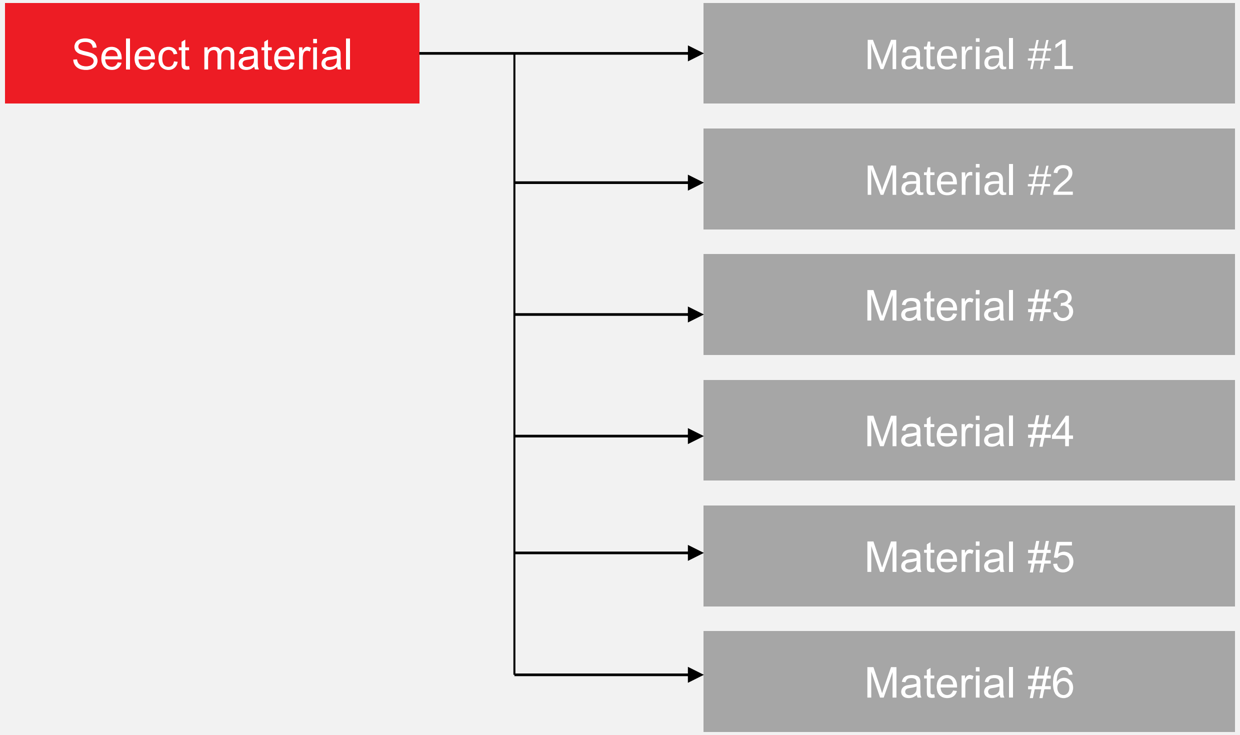
EVEN MORE OPPORTUNITIES

“Parallel chains”: For each new meshlet, unpack data, apply displacement, animate/pose



EVEN MORE OPPORTUNITIES

“Function calls”: Ray-tracing and materials, anyone?



AMD
GPUOpen

Work Graphs

The next generation of GPU programmability

#GDC2024

WHAT IF …

1

The GPU could decide **when/what to launch?**

2

The GPU would **allocate/free memory** for you?

3

The GPU could do **all sorts of black-box things** you can't influence but which help performance ☺?

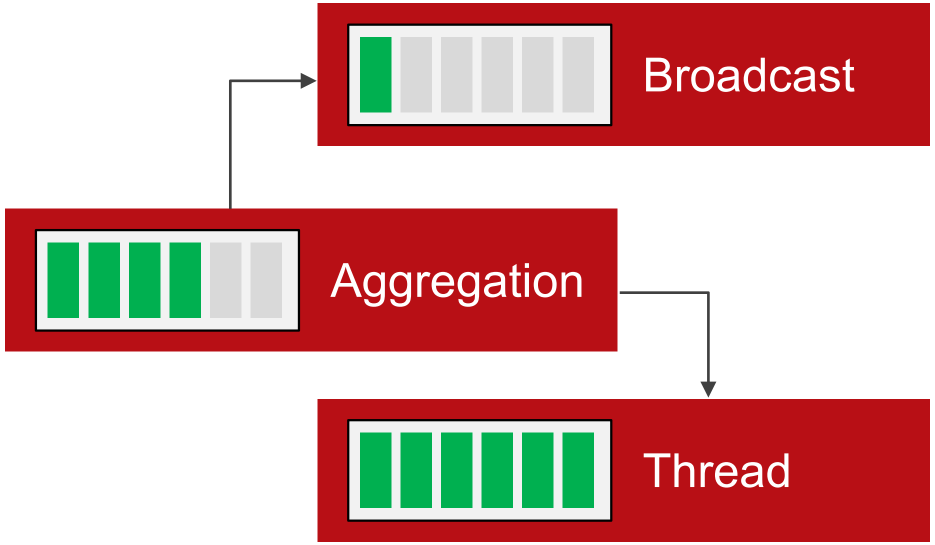
What if you could use this today? 🤯

(You actually can. No, seriously, get the driver and try it!)

SAY WHAAAT?

GPU work graph is…

- a data flow model
- Work moves from node to node
  in form of small "work items" (think: a struct)
- Work items get "queued up"
- Once enough work is pending,
  the GPU launches a dispatch



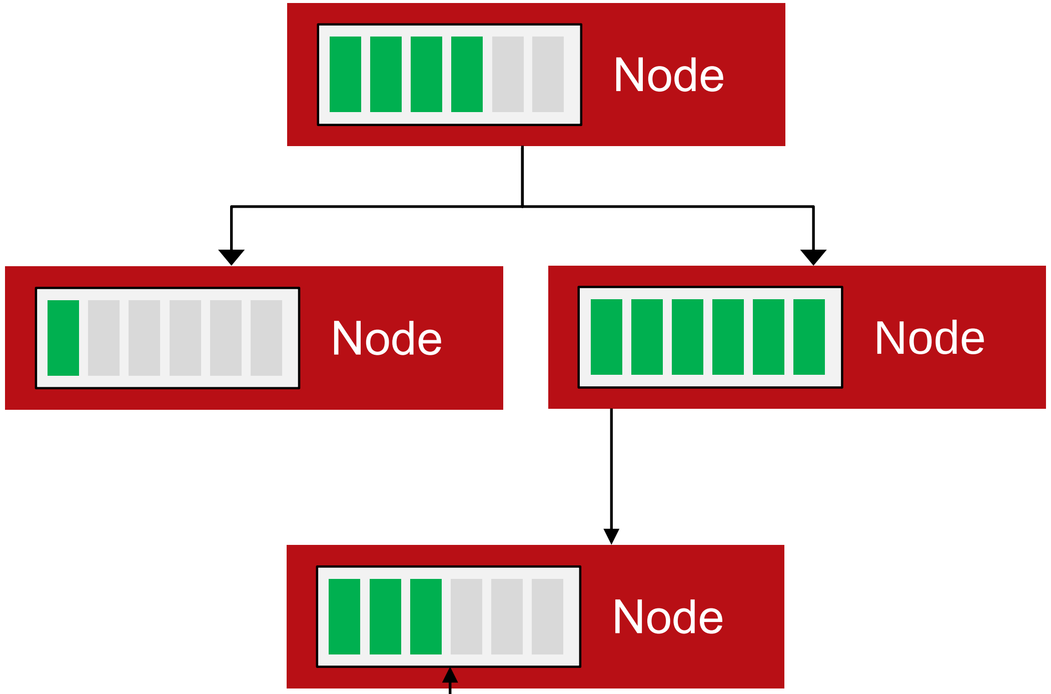
WORK GRAPHS IN A NUTSHELL

Nodes connected with **edges**

Each node has a virtual **queue**

Nodes launch as soon as **“enough” work** waits for them

- Enough depends on the GPU, driver, …
- Runtime can merge/fuse nodes, reorder outputs, sort, etc.



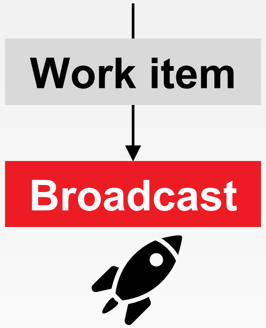
Scheduler launches dispatch to consume the data

LAUNCH WHAT?

You can select how things launch. Work items can …

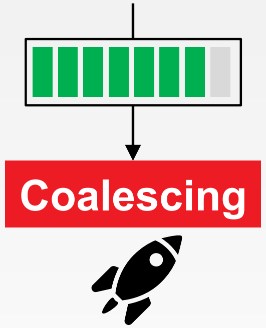
1

Trigger dispatch
(“broadcasting”)



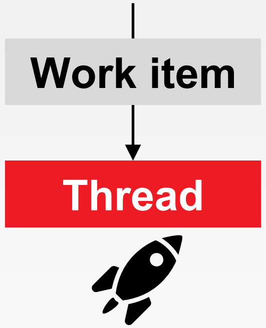
2

Be aggregated
(“coalescing”)



3

Be treated as independent launches
(“thread”)

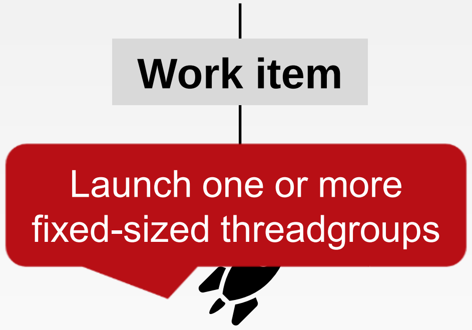


LAUNCH WHAT?

You can select how things launch. Work items can …

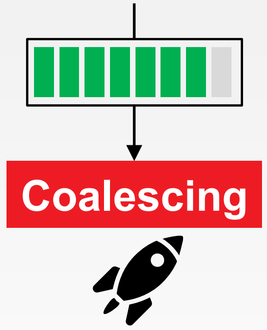
1

Trigger dispatch
(“broadcasting”)



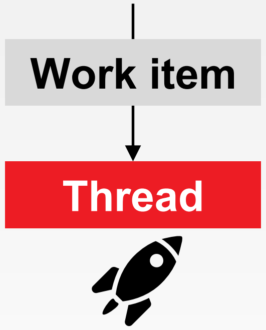
2

Be aggregated
(“coalescing”)



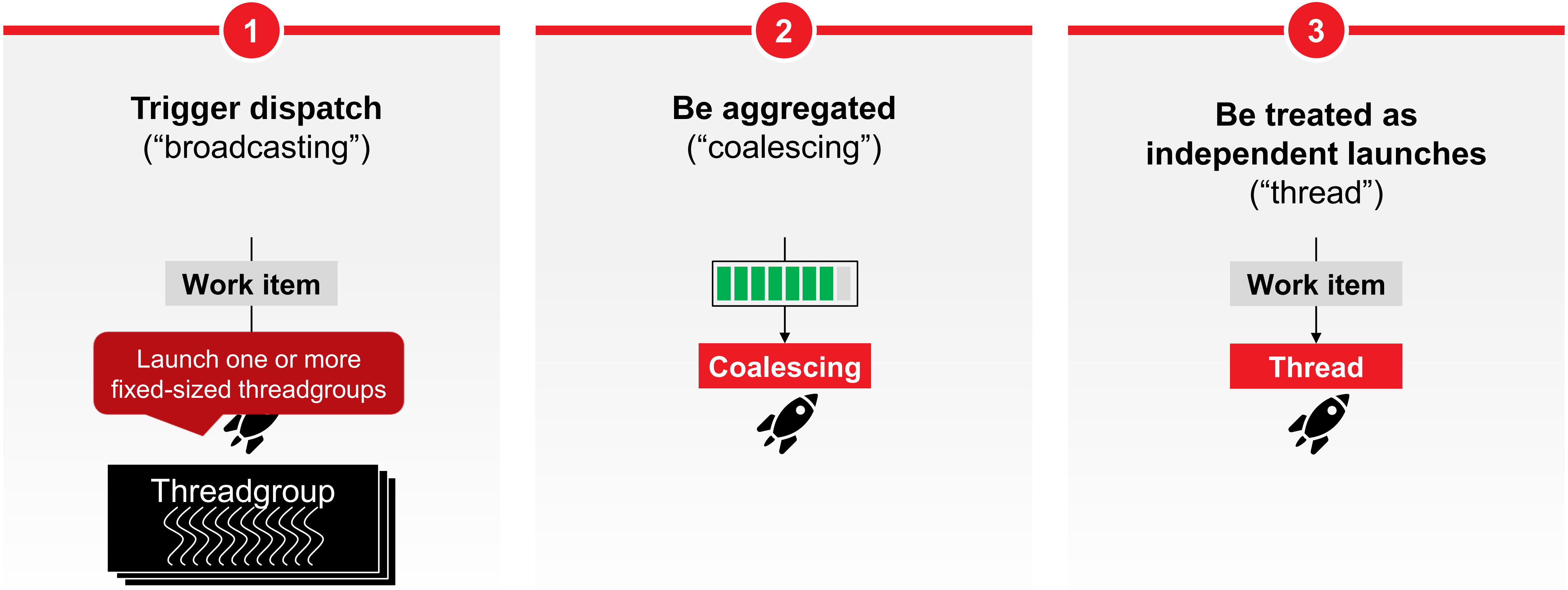
3

Be treated as independent launches
(“thread”)



LAUNCH WHAT?

You can select how things launch. Work items can …

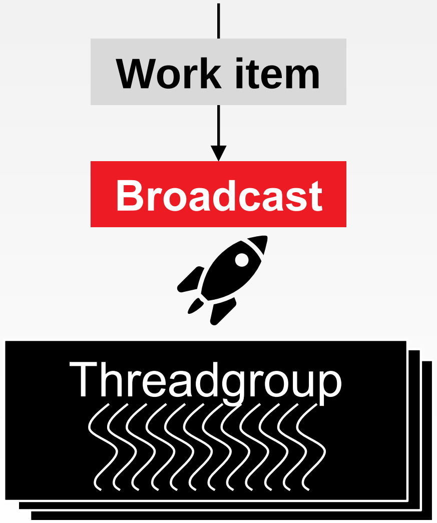


LAUNCH WHAT?

You can select how things launch. Work items can …

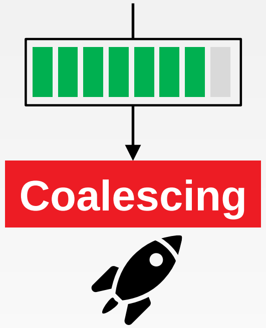
1

Trigger dispatch
(“broadcasting”)



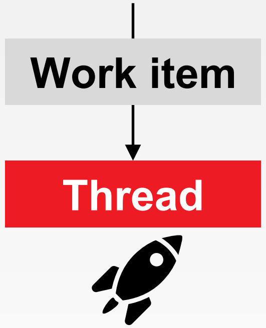
2

Be aggregated
(“coalescing”)



3

Be treated as independent launches
(“thread”)

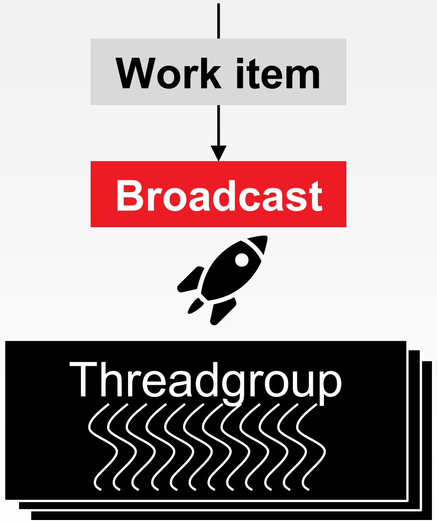


LAUNCH WHAT?

You can select how things launch. Work items can …

1 Trigger dispatch

(“broadcasting”)



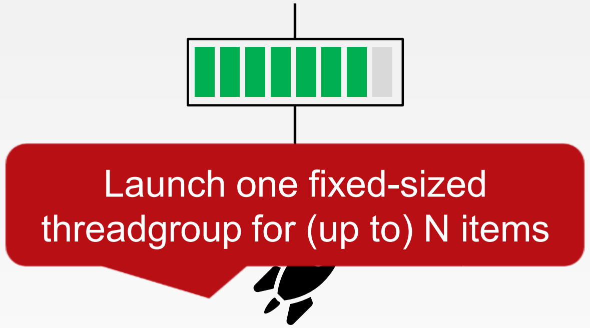
Work item

Broadcast

Threadgroup

2 Be aggregated

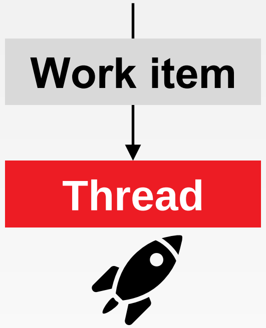
(“coalescing”)



Launch one fixed-sized threadgroup for (up to) N items

3 Be treated as independent launches

(“thread”)



Work item

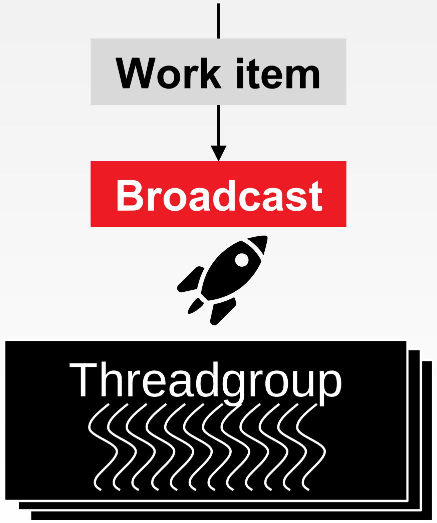
Thread

LAUNCH WHAT?

You can select how things launch. Work items can …

1

Trigger dispatch
(“broadcasting”)



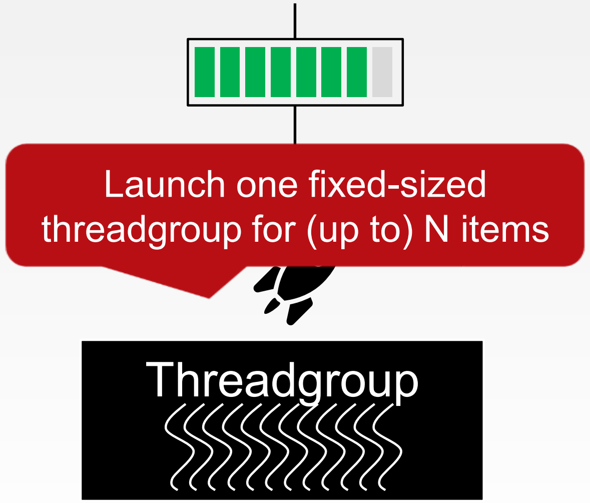
Work item

Broadcast

Threadgroup

2

Be aggregated
(“coalescing”)

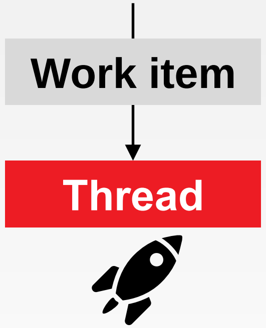


Launch one fixed-sized threadgroup for (up to) N items

Threadgroup

3

Be treated as independent launches
(“thread”)



Work item

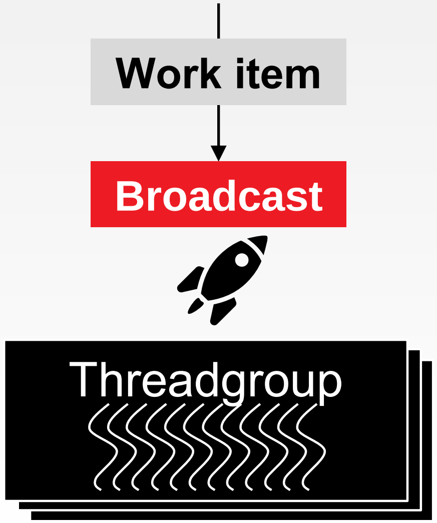
Thread

LAUNCH WHAT?

You can select how things launch. Work items can …

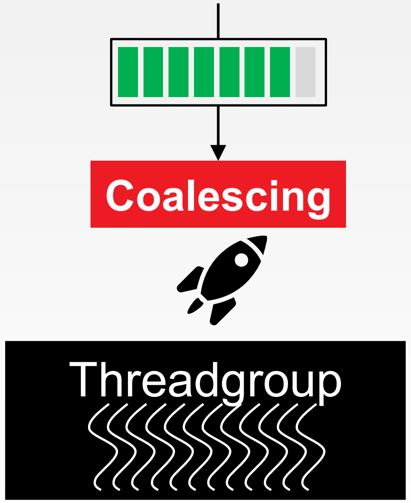
1

Trigger dispatch
(“broadcasting”)



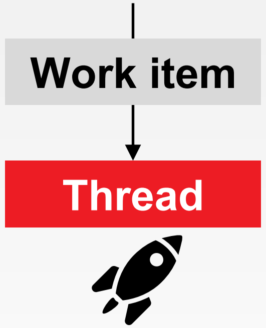
2

Be aggregated
(“coalescing”)



3

Be treated as independent launches
(“thread”)

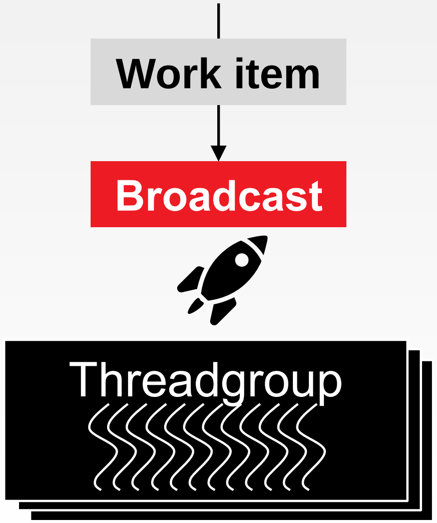


LAUNCH WHAT?

You can select how things launch. Work items can …

1

Trigger dispatch
(“broadcasting”)

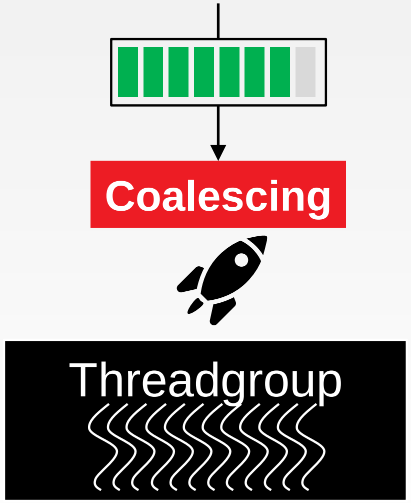

Work item

Broadcast

Threadgroup

2

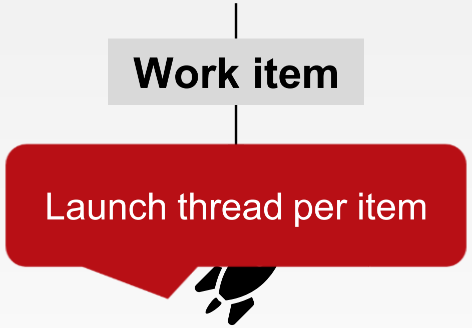
Be aggregated
(“coalescing”)


Coalescing

Threadgroup

3

Be treated as independent launches
(“thread”)


Work item

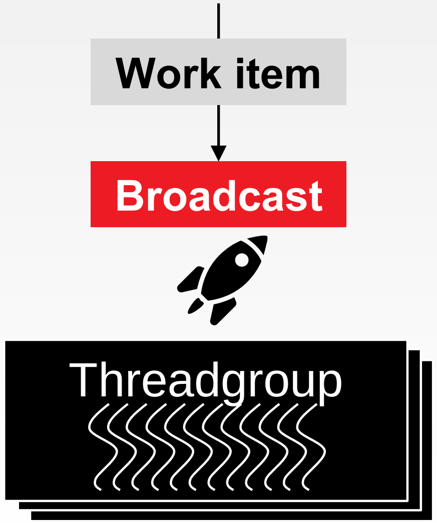
Launch thread per item

LAUNCH WHAT?

You can select how things launch. Work items can …

1

Trigger dispatch
(“broadcasting”)



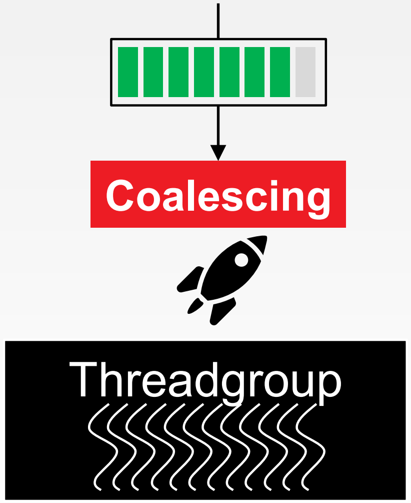
Work item

Broadcast

Threadgroup

2

Be aggregated
(“coalescing”)

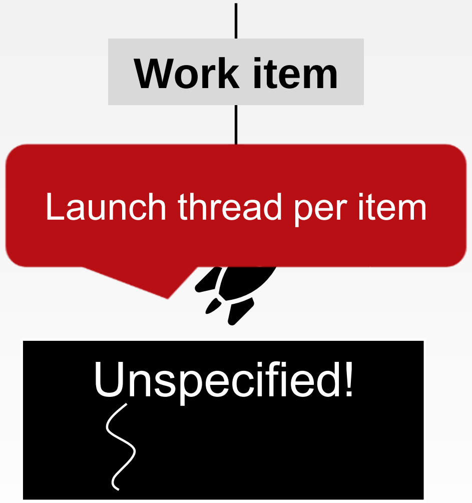


Coalescing

Threadgroup

3

Be treated as independent launches
(“thread”)



Work item

Launch thread per item

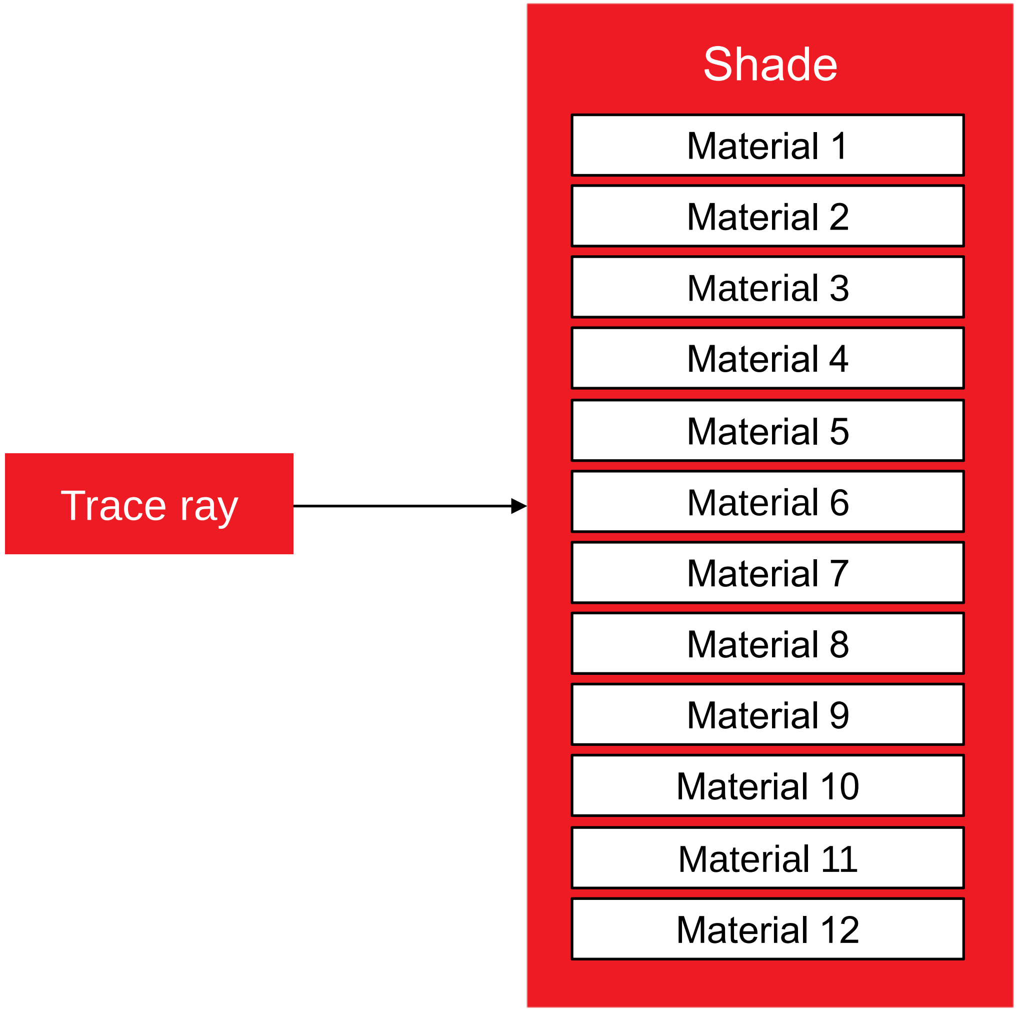
Unspecified!

WORK GRAPHS IN A NUTSHELL

Nodes can be “node arrays”

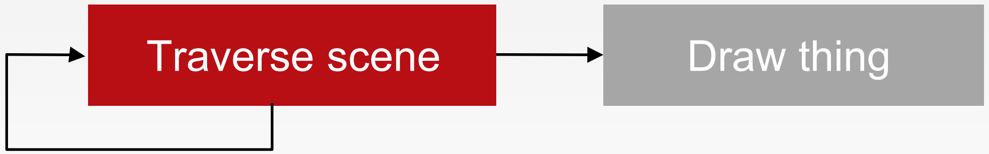
Uniform input type

Allows you to select “one of many” easily
(can vary per lane, for example)



WORK GRAPHS IN A NUTSHELL

Self-recursion is allowed,
but no loops across nodes



Total depth and expansion is limited

[max depth=32, expansion: 1:32768 unless thread launch (i.e. 32 KiBx32 = 1MiB)]

SYNTAX

Plain old HLSL
Extra annotations for a function – that’s it!

```
[Shader("node")]
[NodeLaunch("broadcasting")]
[NodeMaxDispatchGrid(65535, 1, 1)]
[NodeIsProgramEntry] // optional
[NumThreads(TRANSF_NUM_THREADS, 1, 1)]
void TriangleFetchAndTransform(
    uint    WorkloadIndex : SV_GroupID,
    uint    SIMDLaneIndex : SV_GroupIndex,

    // Input record that contains the dispatch grid size.
    // Set up by the application.
    DispatchNodeInputRecord<DrawRecord> launchRecord,
```

Turns a function into a node

SYNTAX

Calling other nodes looks like message passing
Allocate a record, fill it out, done

```
ThreadNodeOutputRecords<RasterizeRecord> rasterRecord =
    triangleOutput[triangleBin].GetThreadNodeOutputRecords(allocateRecordForThisThread);

if (allocateRecordForThisThread)
{
    rasterRecord.Get().tri = StoreTriangleState(ts);
}

rasterRecord.OutputComplete();
```

Wrote payload and “send” it

CAN I BEAT THE BLACK BOX?

Yes, sometimes. Heroic programming!

Persistent **threads**

Custom **memory management**

Low-level **synchronization** tricks

Work graphs make all of this accessible, easier to compose, give the runtime “optimization freedom” and enable new features down the line

CAN I BEAT THE BLACK BOX?

Yes, sometimes. Heroic programming!

Persistent **threats**

Custom **memory management**

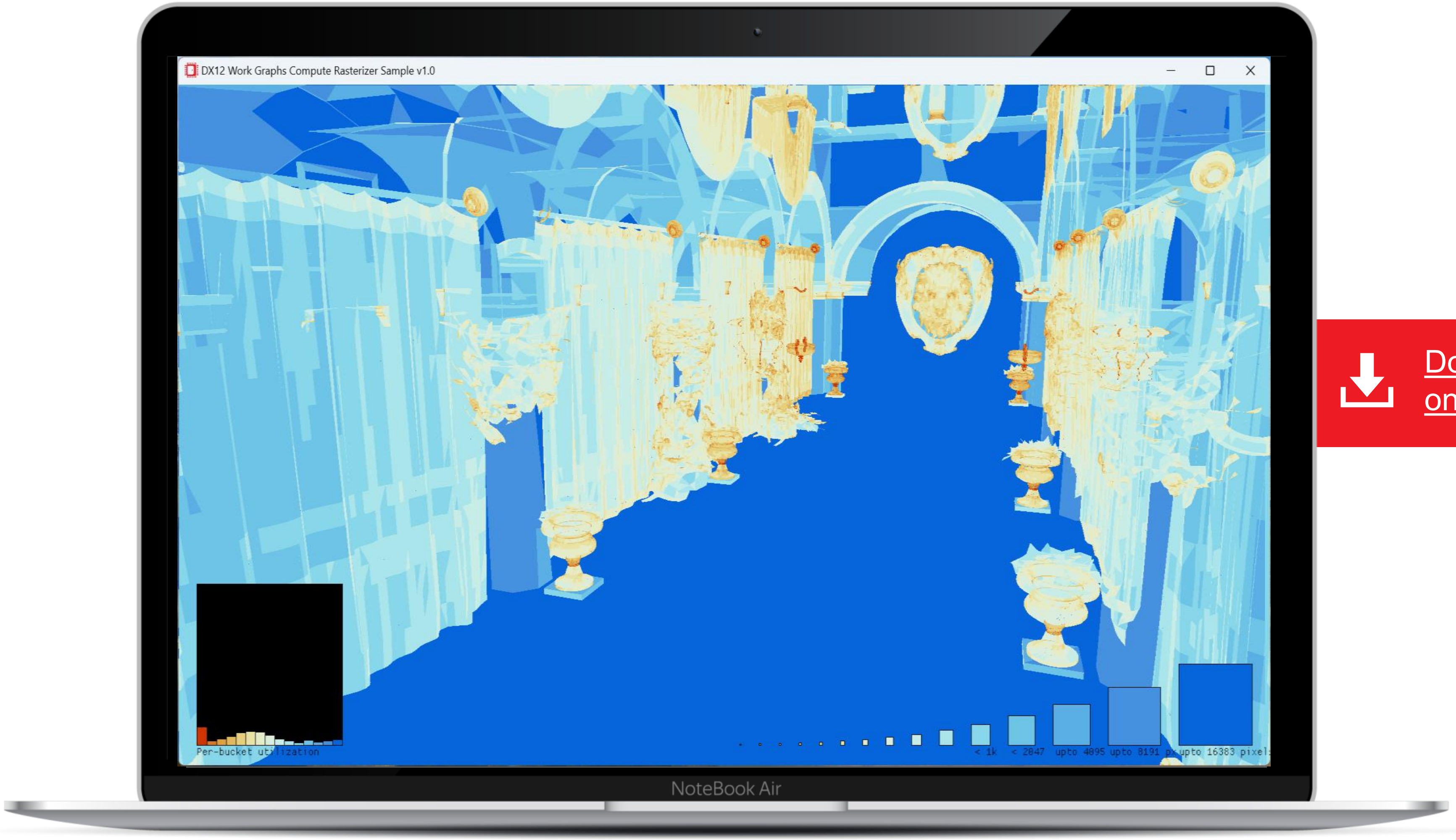
Low-level **synchronization** tricks

Work graphs make all of this accessible, easier to compose, give the runtime “optimization freedom” and enable new features down the line

Practical applications

Work graphs in the wild!

COMPUTE RASTERIZATION

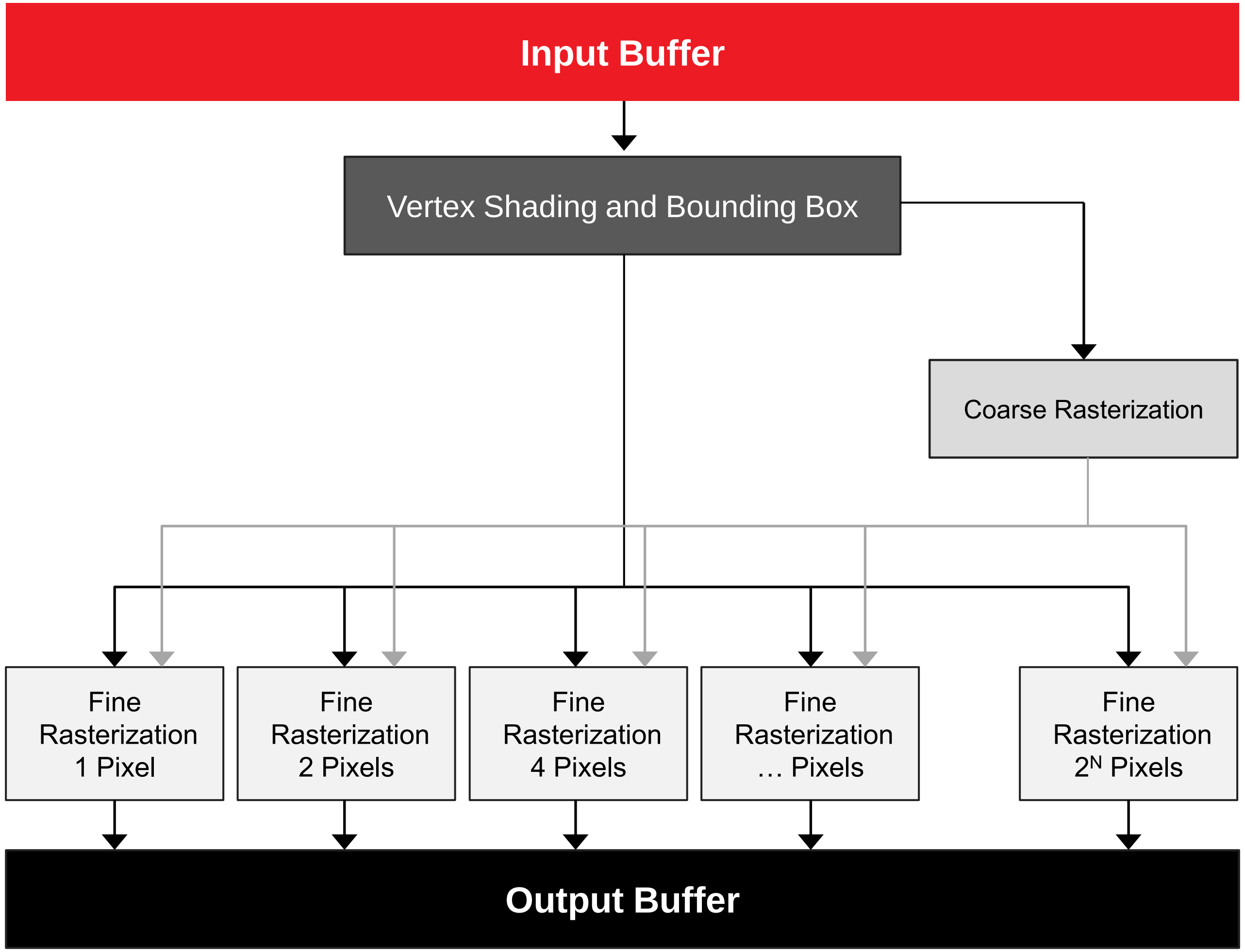


Download today on GPUOpen

USE CASE: COMPUTE RASTERIZATION

Computer rasterizer: Needs to deal with varying triangle sizes

- Best performance: **Sort by size**
- One bucket per size, holding potentially all triangles?
- Extra barrier between producer/consumer

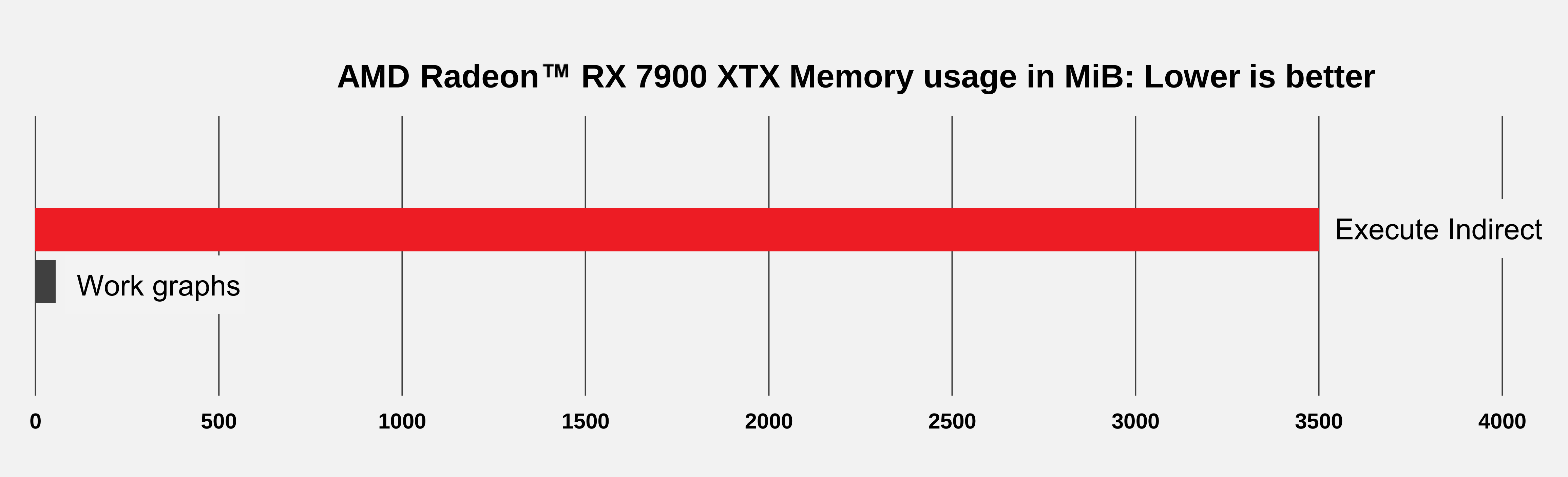

Input Buffer
Vertex Shading and Bounding Box
Coarse Rasterization
Fine Rasterization 1 Pixel
Fine Rasterization 2 Pixels
Fine Rasterization 4 Pixels
Fine Rasterization … Pixels
Fine Rasterization 2^N Pixels
Output Buffer

USE CASE: COMPUTE RASTERIZATION

Work graphs vs. ExecuteIndirect

Reduced memory usage: 3500 MiB → 55 MiB (🤯)

Slightly improved performance



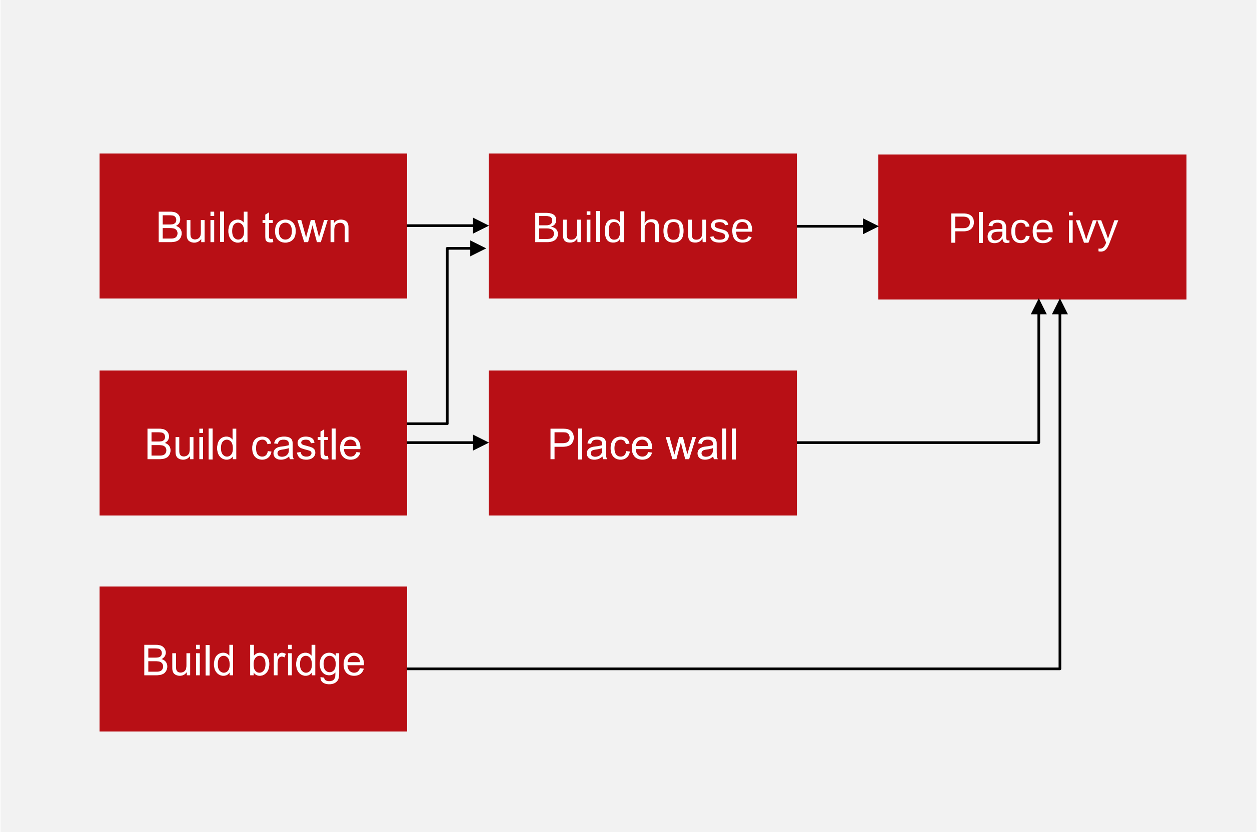
AMD Radeon™ RX 7900 XTX Memory usage in MiB: Lower is better

Measured on AMD Radeon RX 7900 XTX, 2024-02-26, internal driver

USE CASE: PROCEDURAL CONTENT

Procedural content creation can be implemented through “node graphs” (see Blender®, Houdini™, etc.)

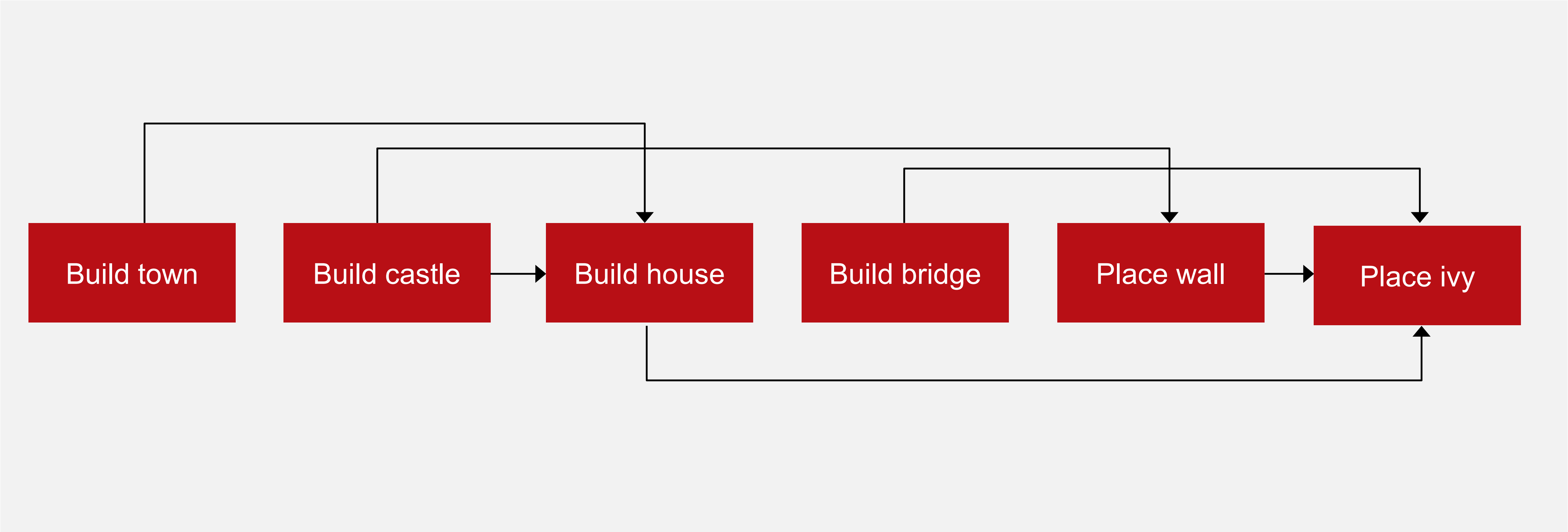
Complex decision trees make it hard to run on execute indirect (branch/merge – what’s the worst-case ivy count?)



USE CASE: PROCEDURAL CONTENT

Don’t do it this way!

ExecuteIndirect requires topological graph sort, allocating multiple output buffers, dependency tracking, etc.



PROCEDURAL ENRICHMENT

GDC 2024

ADVANCED GRAPHICS SUMMIT

GPU WORK GRAPHS
WELCOME TO THE FUTURE OF GPU PROGRAMMING

Matthäus Chajdas (AMD)
Shawn Hargreaves (Microsoft®)

AMD
together we advance_game dev

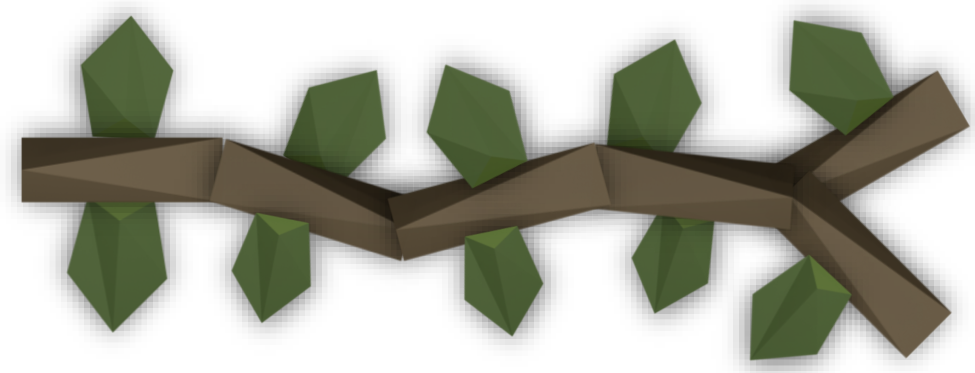
Live demo

All generation and all rendering in every frame

((•)) LIVE ((•))

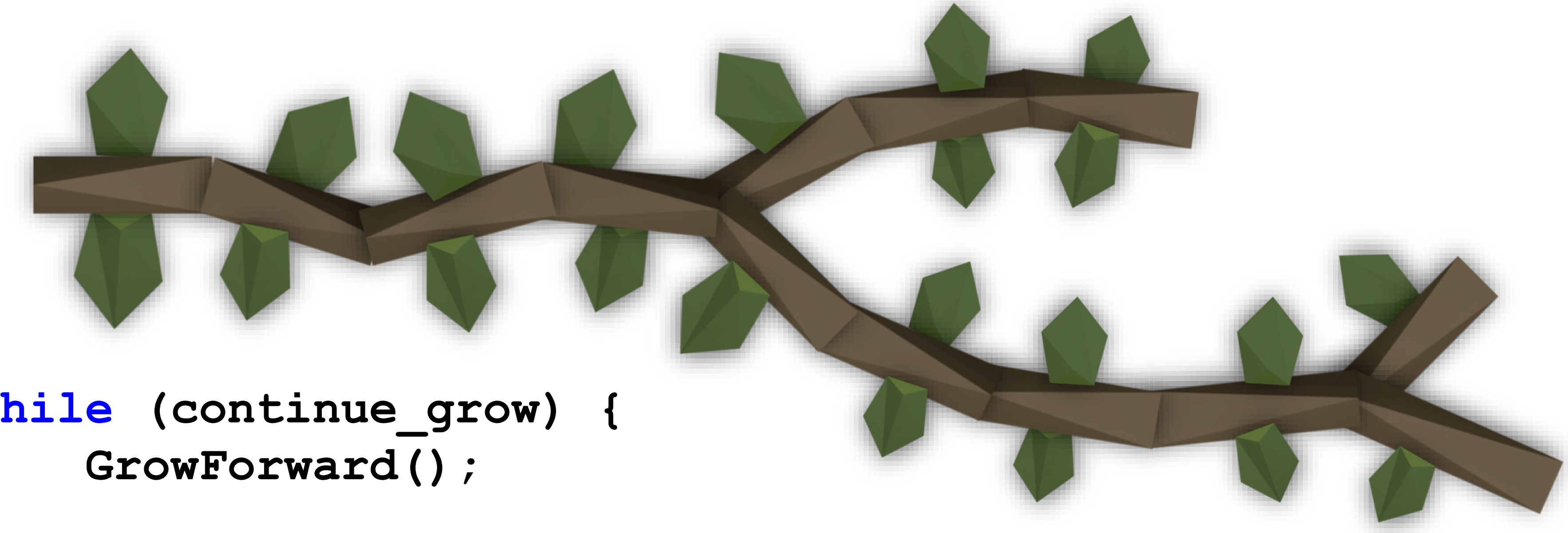
“THE BRIDGE”

PROCEDURAL CONTENT DEMO



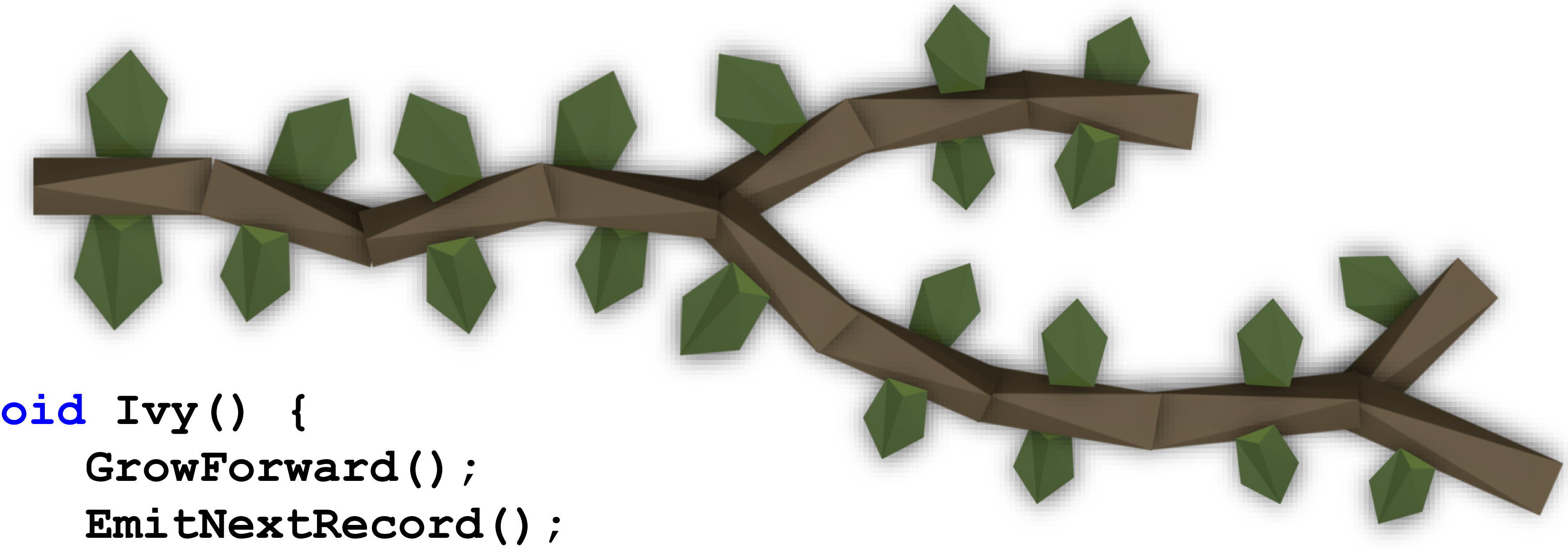
```
while (continue_grow) {
    GrowForward();
}
```

PROCEDURAL CONTENT DEMO

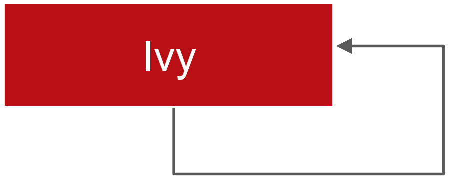


```
while (continue_grow) {
    GrowForward();

    if (forked) {
        // TODO: figure this out properly;
        // maybe use a stack or something
    }
}
```

PROCEDURAL CONTENT DEMO

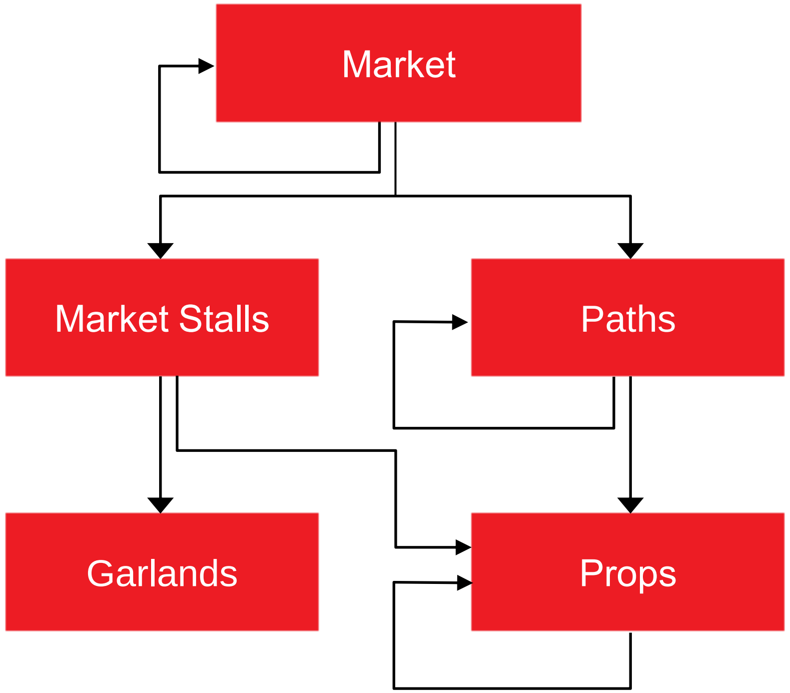


```
void Ivy() {
    GrowForward();
    EmitNextRecord();

    if (forked) {
        EmitNextRecord();
    }
}
```

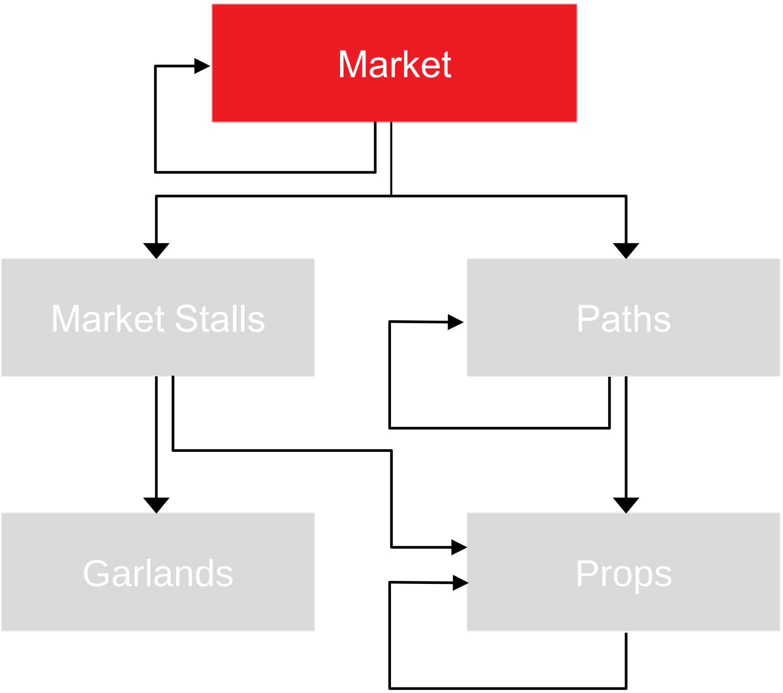


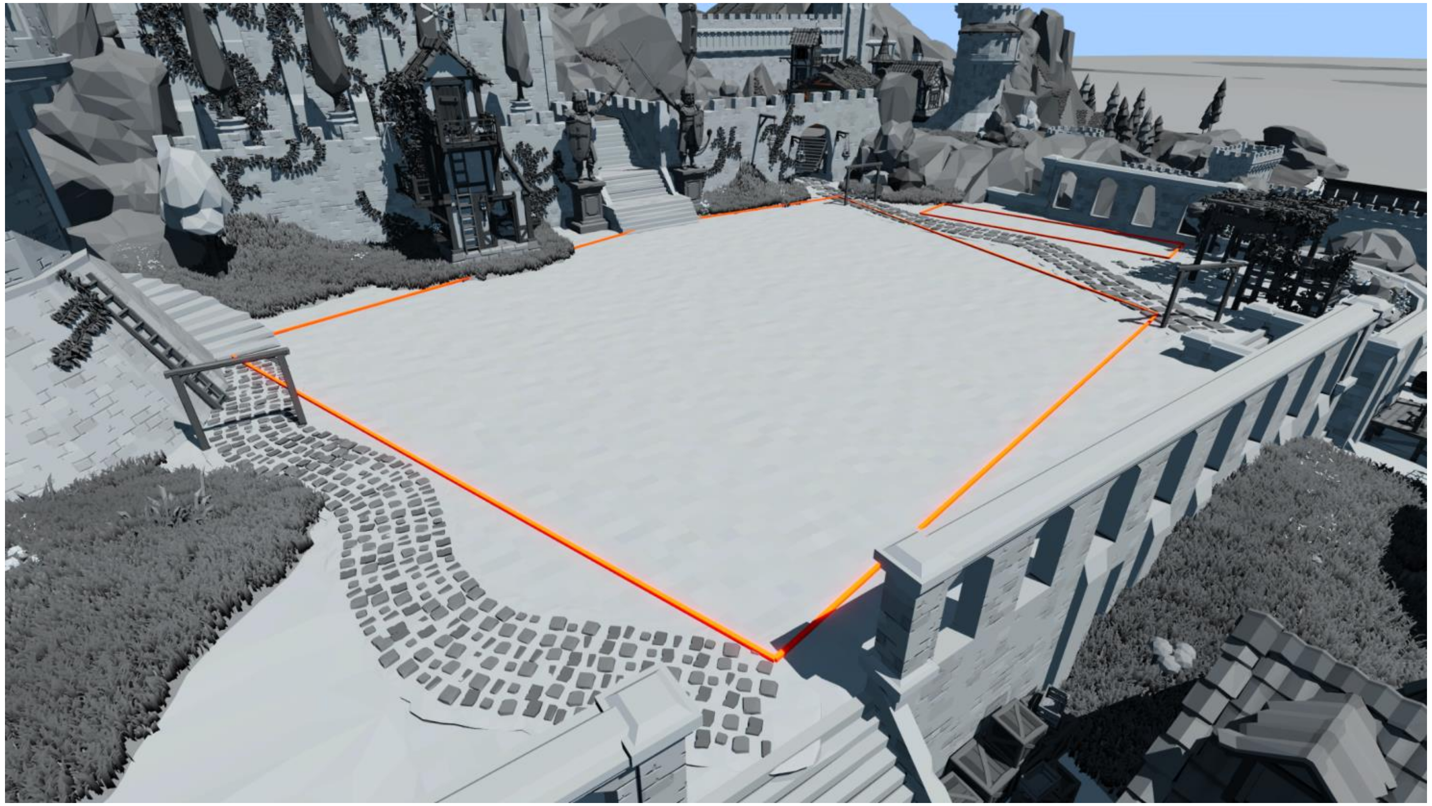
“THE MARKET”

PROCEDURAL CONTENT DEMO

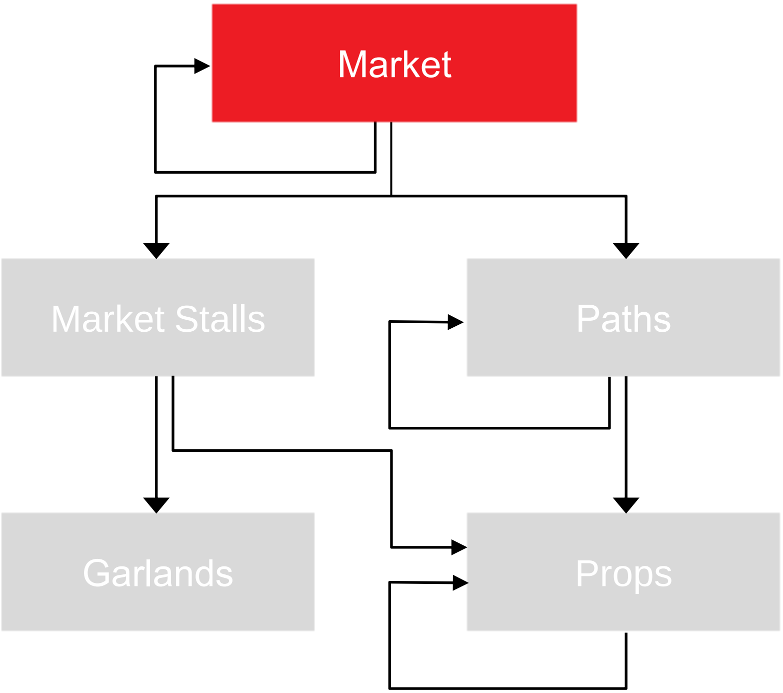


PROCEDURAL CONTENT DEMO





PROCEDURAL CONTENT DEMO



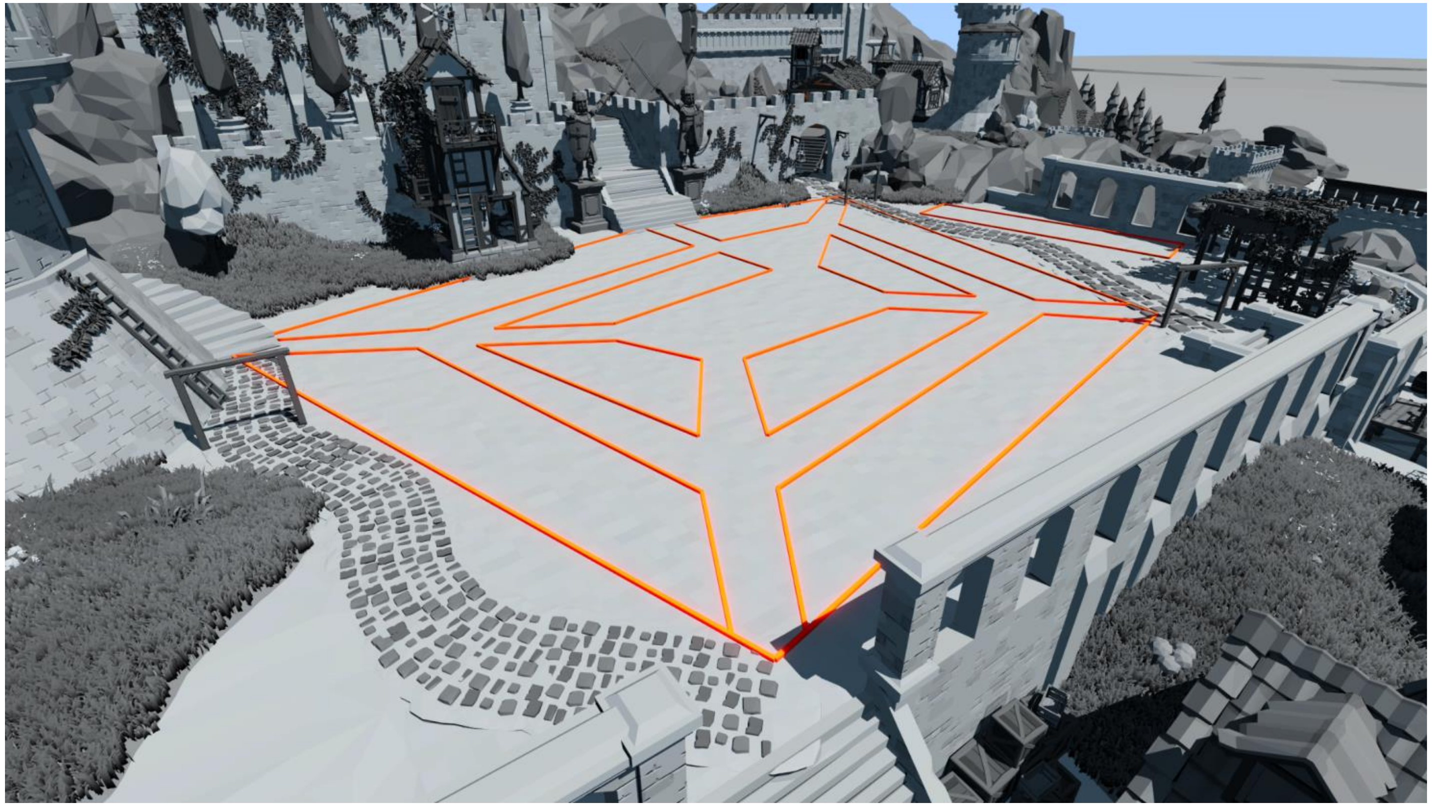
Market

Market Stalls

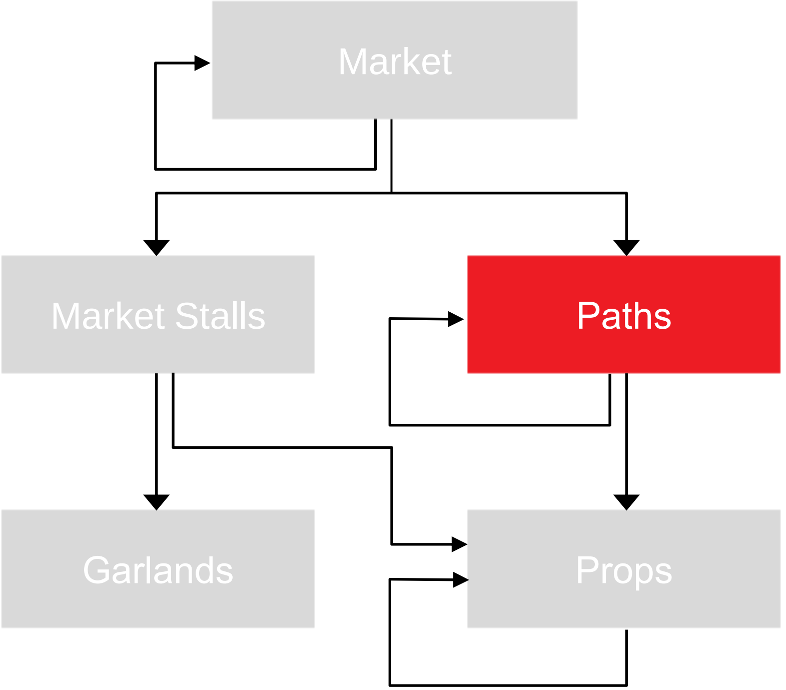
Paths

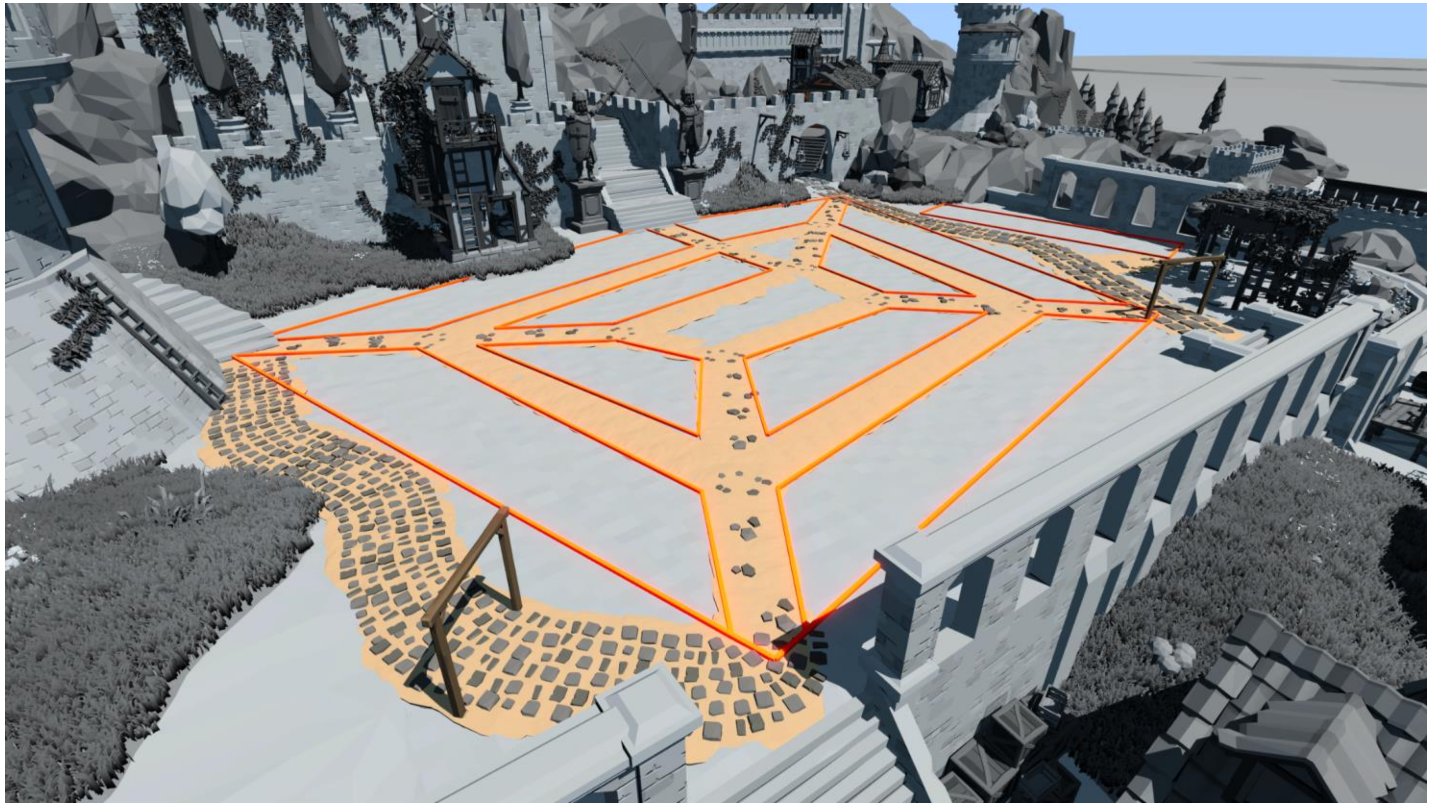
Garlands

Props

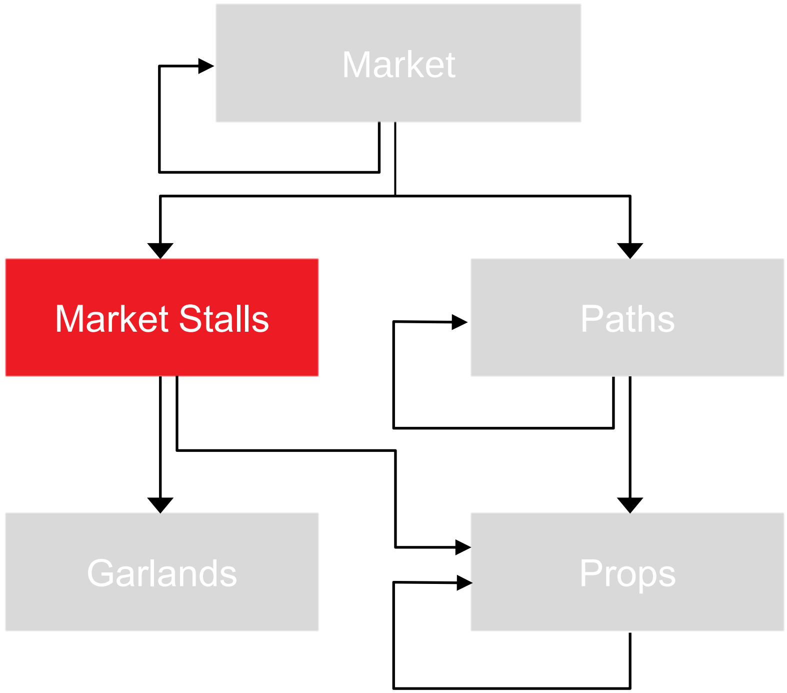


PROCEDURAL CONTENT DEMO

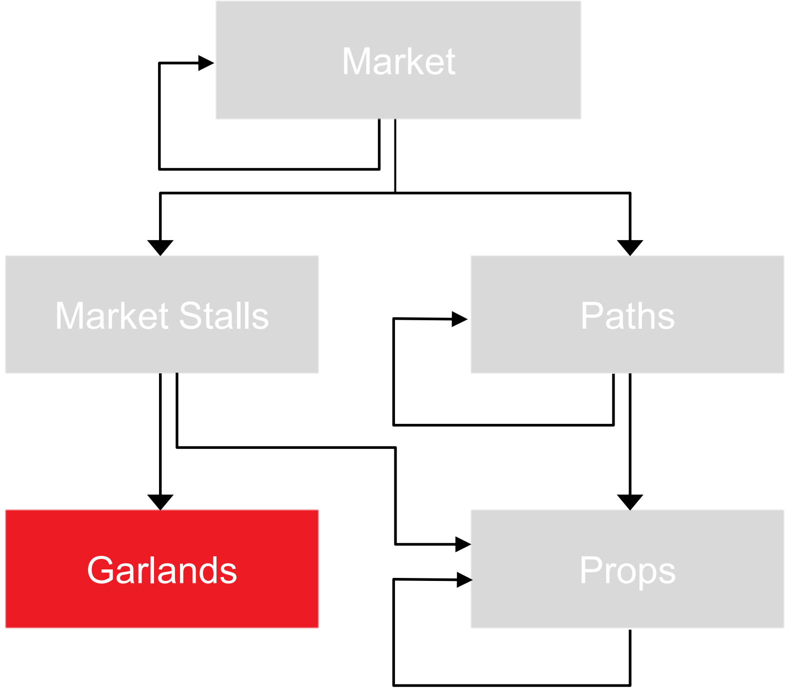


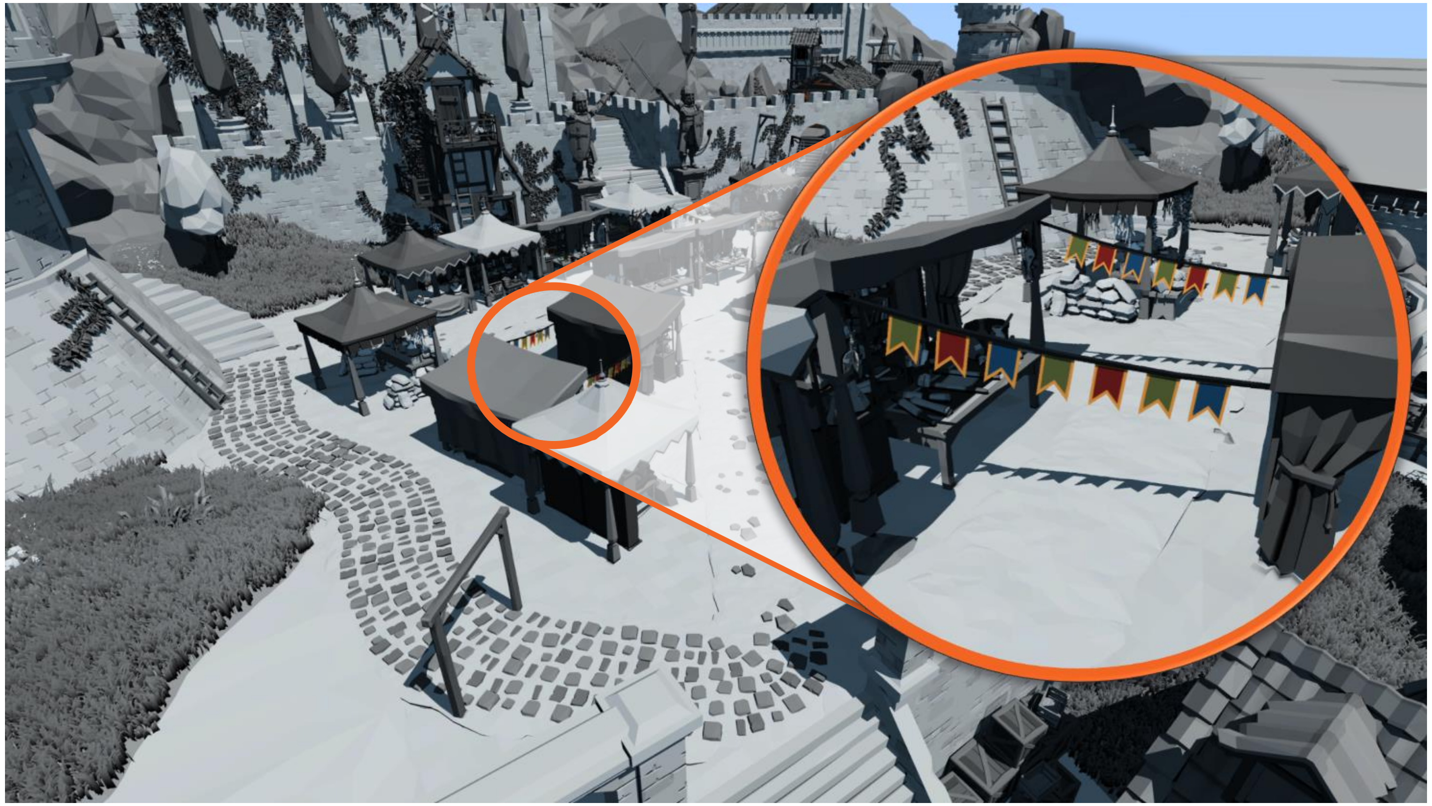


PROCEDURAL CONTENT DEMO

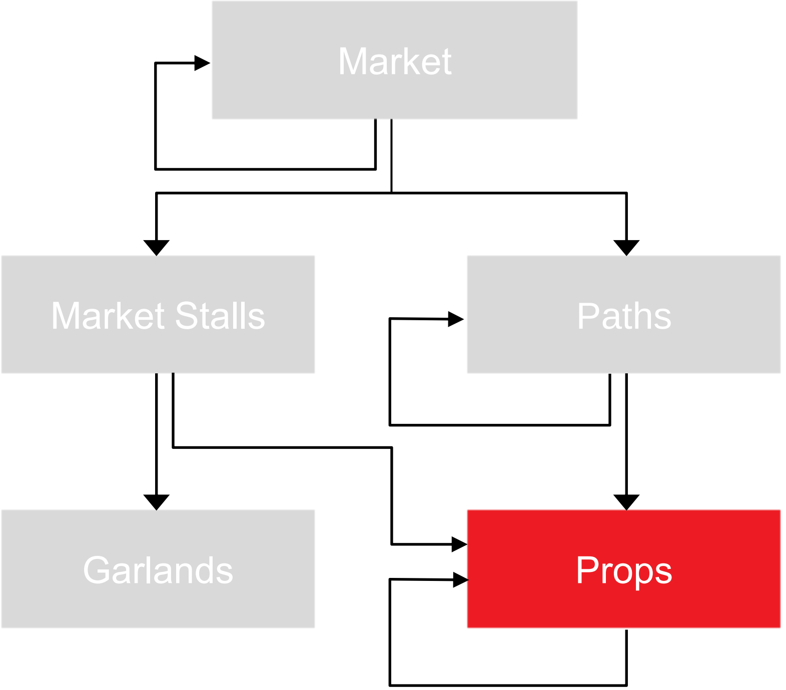


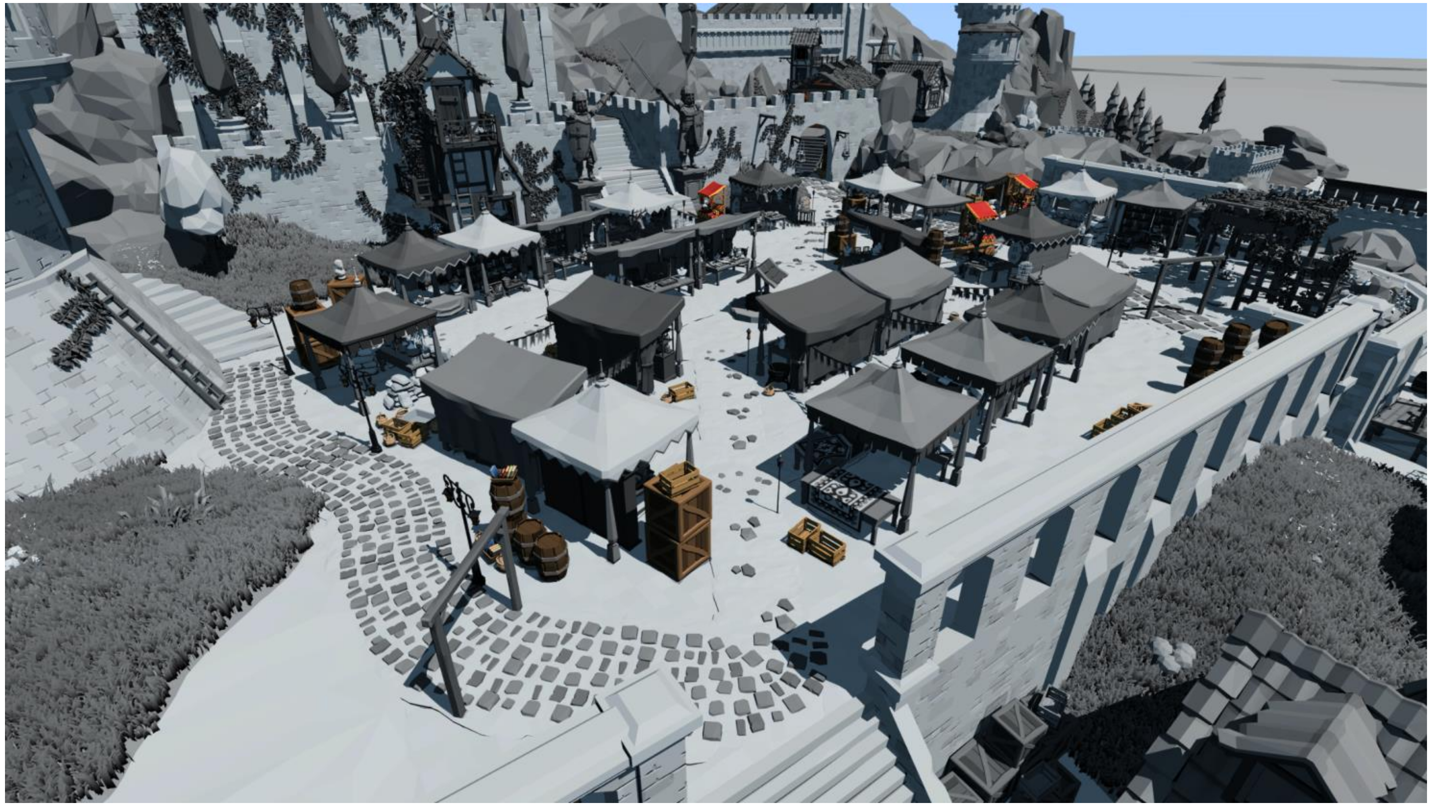
PROCEDURAL CONTENT DEMO





PROCEDURAL CONTENT DEMO





THE MEADOW

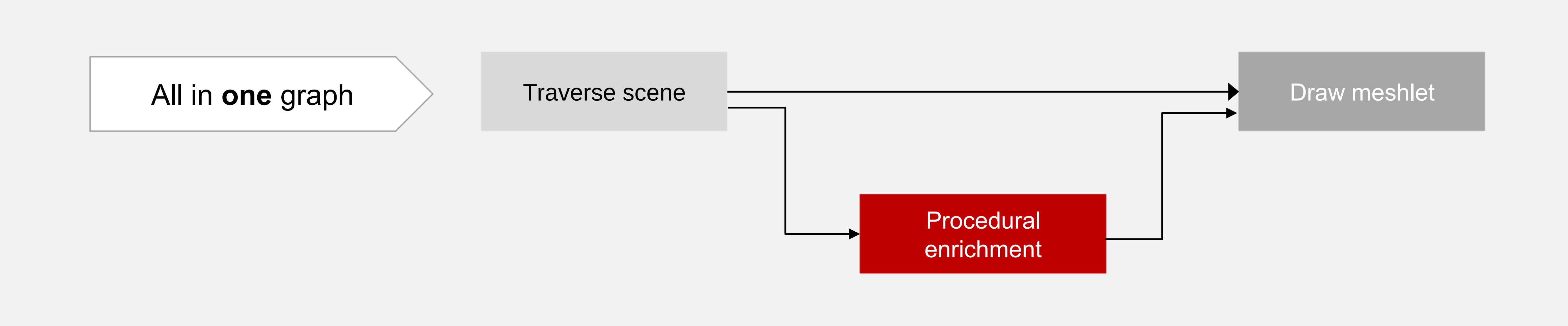
⚠ PREVIEW ⚠

MESH NODES

Preview feature announcement: Mesh nodes

Draw “inside” the work graph using “mesh nodes”

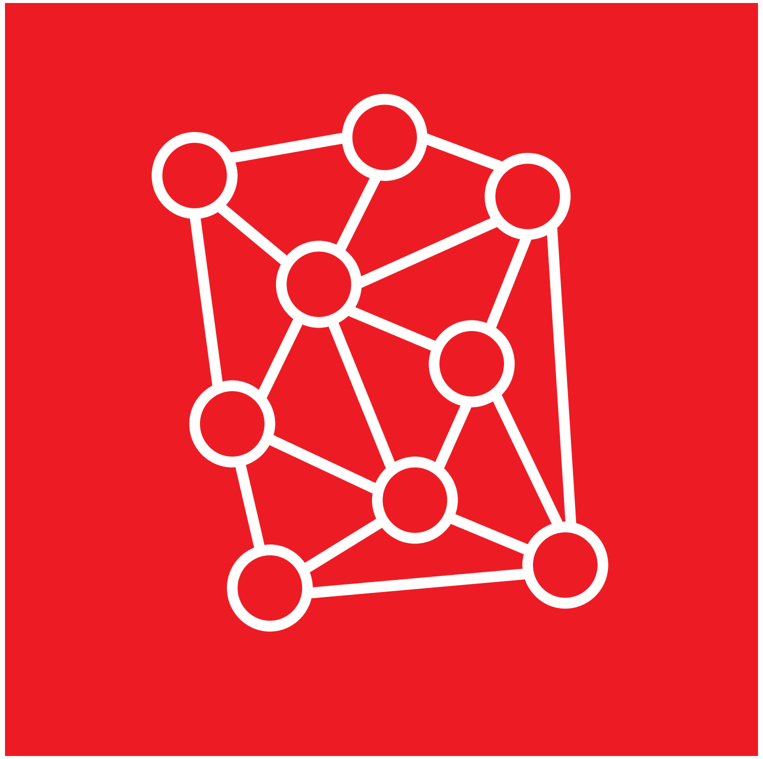
Enables fully compute-driven scene traversal **(with PSO switching)**



[AMD Official Use Only – General]

PREVIEW

MESH NODES



Mesh Nodes: Feed into a **mesh shader pipeline**

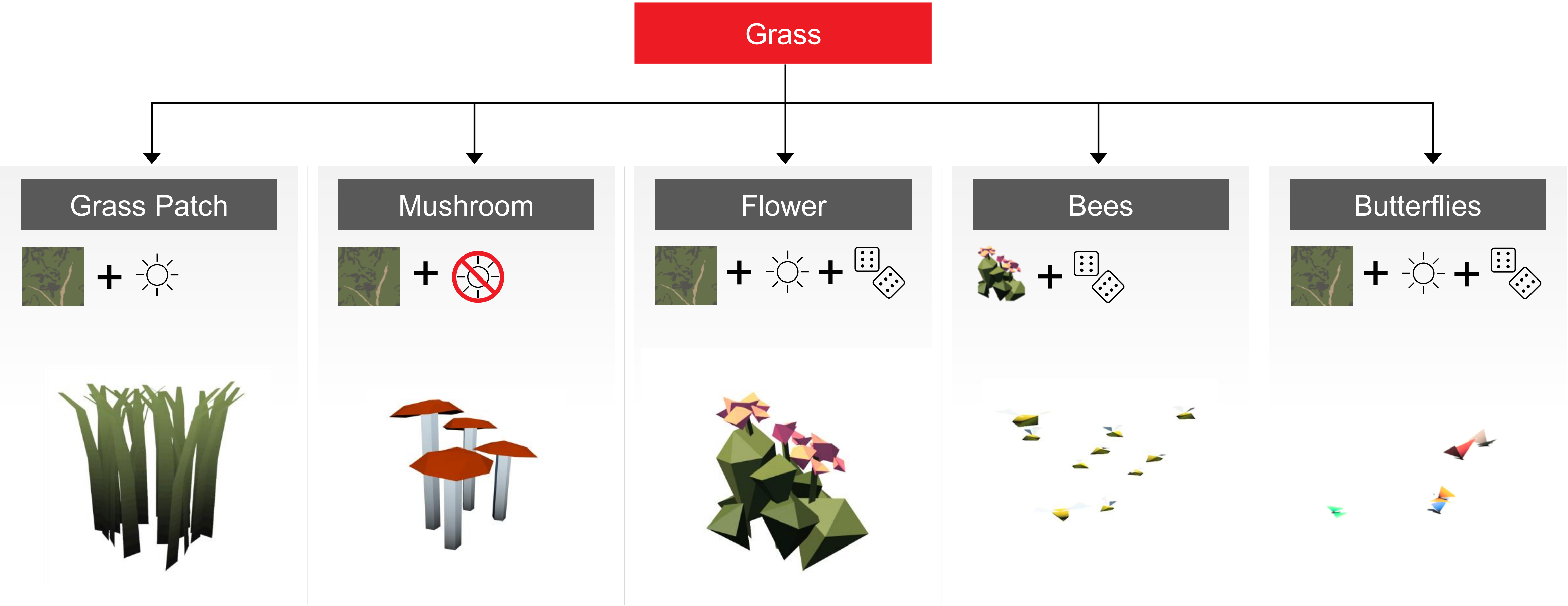
Work graph acts like an **amplification shader** on steroids

Runtime ensures PSO switching isn't too expensive

- Will buffer up draw calls per state
- Will optimize state changes
- The more similar the states are, the better – cheapest state change is swapping out shaders only

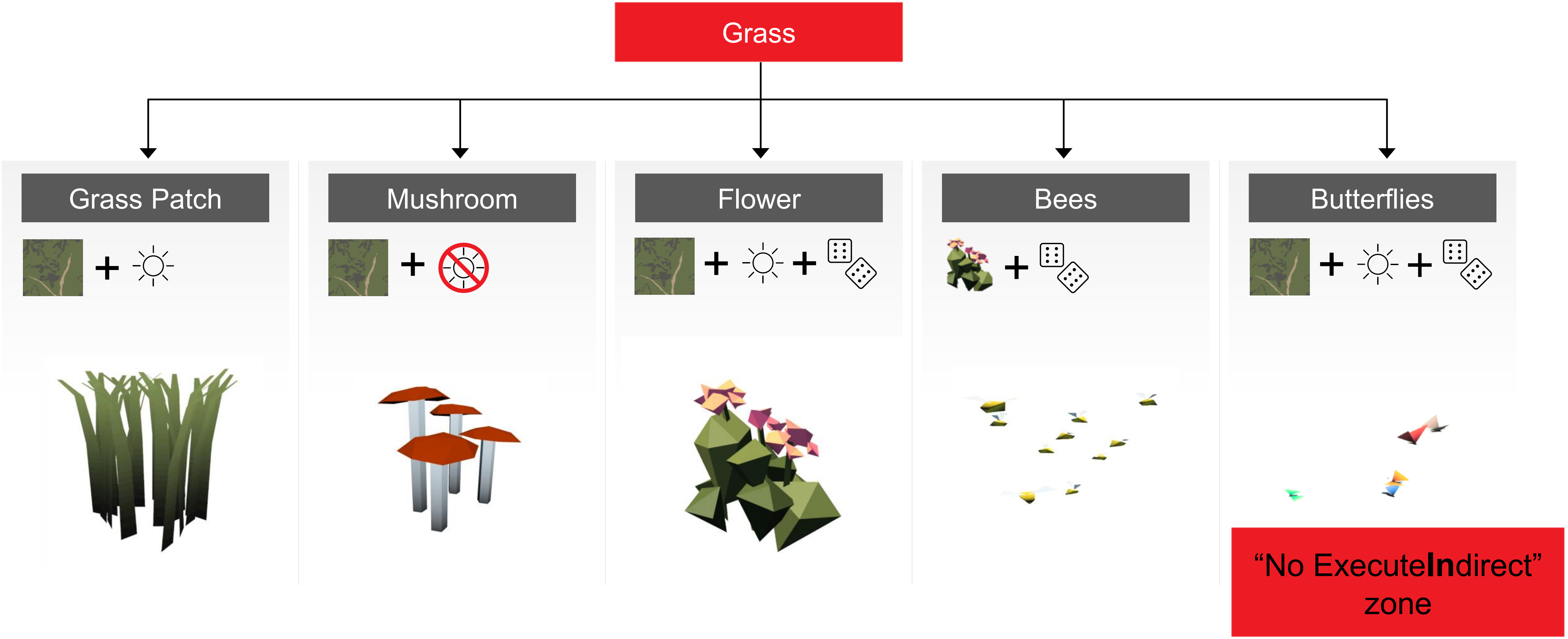
GRASS GENERATION

PREVIEW



PREVIEW

GRASS GENERATION



“No ExecuteIndirect” zone

PREVIEW

STATS FOR THE DEMO

37
nodes

+9
mesh nodes

6.6K
draws/frame

13M
triangles/frame

196 MiB
of memory

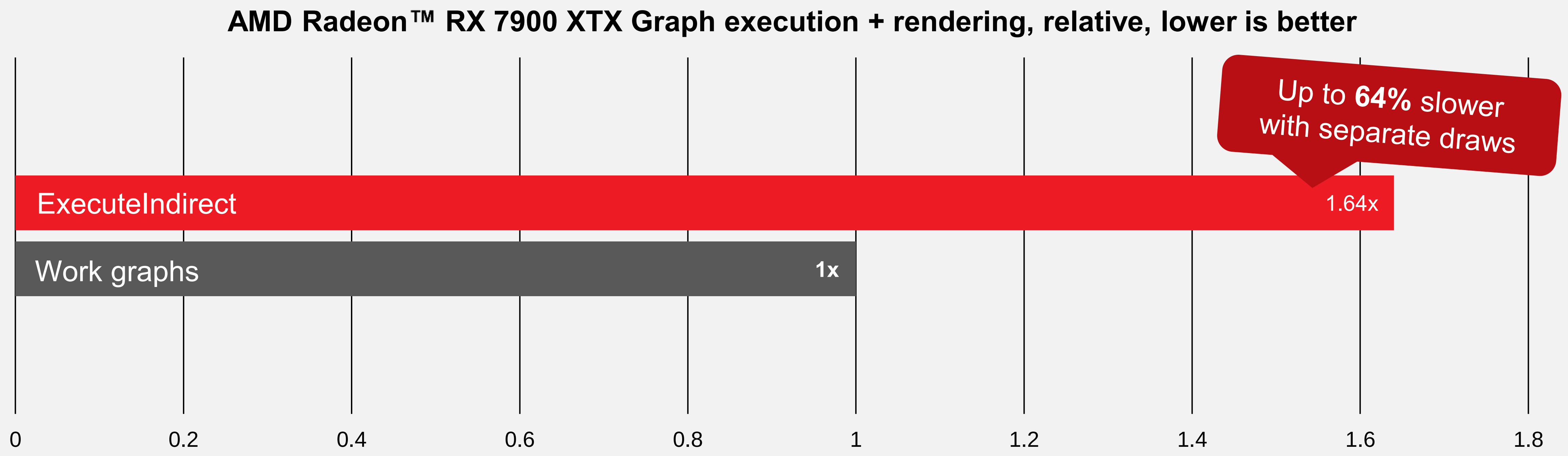
200,000
work items

Everything ran all the time in **every** frame

MESH NODES: PERFORMANCE

Work graphs vs. ExecuteIndirect: Super early numbers!

⚠ PREVIEW ⚠



Measured on AMD Radeon RX 7900 XTX, 2024-02-26, internal driver

PERFORMANCE PITFALLS

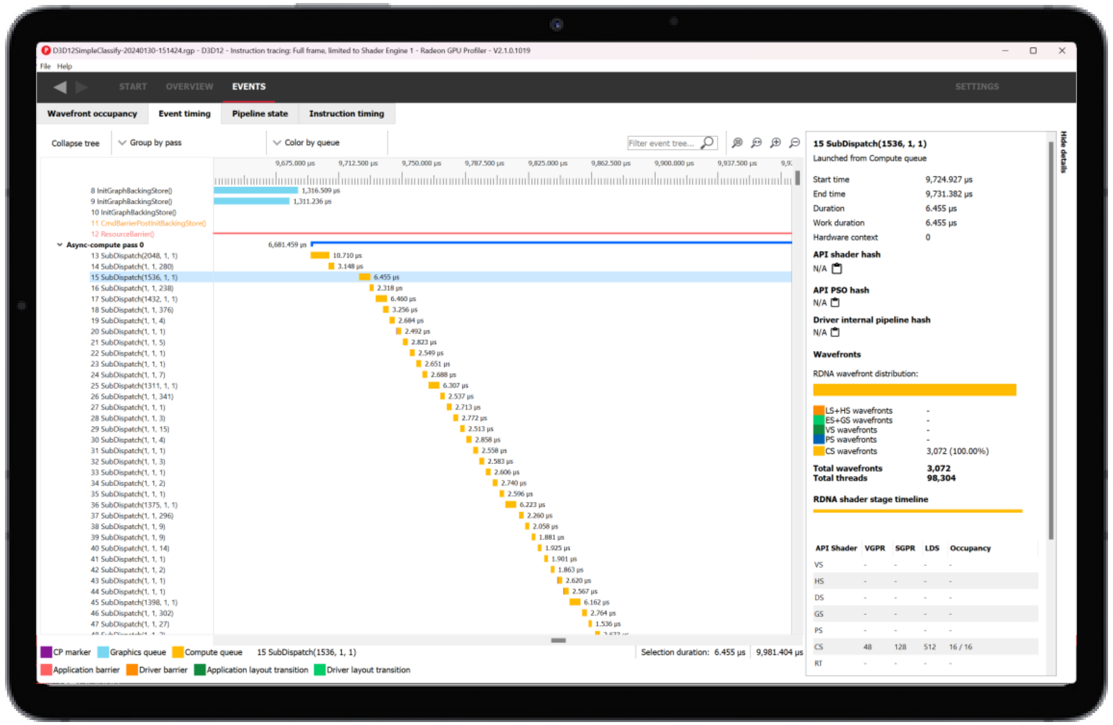
The **smaller the launch, the worse the performance:** Don’t try to go too fine-grained on 1.0 (i.e., make sure that a node accumulates enough work to launch a few thousand threads)

Keep payloads small – ideally, a couple of bytes

Don’t try to synchronize just yet – easy to shoot yourself in the foot, better ideas in the making

Always check how full your input is in coalescing nodes

AMD RADEON™ GPU PROFILER SUPPORT



```
8 InitGraphBackingStore()              1,316.509 μs
9 InitGraphBackingStore()              1,311.236 μs
10 InitGraphBackingStore()
11 CmdBarrierPostInitBackingStore()
12 ResourceBarrier()
Async-compute pass 0                   6,681.459 μs
13 SubDispatch(2048, 1, 1)             10.710 μs
14 SubDispatch(1, 1, 280)              3.148 μs
15 SubDispatch(1536, 1, 1)             6.455 μs
16 SubDispatch(1, 1, 238)              2.318 μs
17 SubDispatch(1432, 1, 1)             6.460 μs
18 SubDispatch(1, 1, 376)              3.256 μs
19 SubDispatch(1, 1, 4)                2.684 μs
20 SubDispatch(1, 1, 1)                2.492 μs
21 SubDispatch(1, 1, 5)                2.823 μs
22 SubDispatch(1, 1, 1)                2.549 μs
23 SubDispatch(1, 1, 1)                2.651 μs
24 SubDispatch(1, 1, 7)                2.688 μs
25 SubDispatch(1311, 1, 1)             6.307 μs
26 SubDispatch(1, 1, 341)              2.537 μs
27 SubDispatch(1, 1, 1)                2.713 μs
28 SubDispatch(1, 1, 3)                2.772 μs
29 SubDispatch(1, 1, 15)               2.513 μs
30 SubDispatch(1, 1, 4)                2.858 μs
31 SubDispatch(1, 1, 1)                2.558 μs
32 SubDispatch(1, 1, 3)                2.583 μs
33 SubDispatch(1, 1, 1)                2.606 μs
34 SubDispatch(1, 1, 2)                2.740 μs
35 SubDispatch(1, 1, 1)                2.596 μs
36 SubDispatch(1375, 1, 1)             6.223 μs
37 SubDispatch(1, 1, 296)              2.260 μs
38 SubDispatch(1, 1, 9)                2.058 μs
39 SubDispatch(1, 1, 9)                1.881 μs
40 SubDispatch(1, 1, 14)               1.925 μs
41 SubDispatch(1, 1, 1)                1.901 μs
42 SubDispatch(1, 1, 2)                1.863 μs
43 SubDispatch(1, 1, 1)                2.620 μs
44 SubDispatch(1, 1, 1)                2.567 μs
45 SubDispatch(1398, 1, 1)             6.162 μs
46 SubDispatch(1, 1, 302)              2.764 μs
47 SubDispatch(1, 1, 27)               1.536 μs

15 SubDispatch(1536, 1, 1)    Selection duration: 6.455 μs    9,981.404 μs
```

15 SubDispatch(1536, 1, 1)

Launched from Compute queue

Start time	9,724.927 μs
End time	9,731.382 μs
Duration	6.455 μs
Work duration	6.455 μs
Hardware context	0

API shader hash

N/A

API PSO hash

N/A

Driver internal pipeline hash

N/A

Wavefronts

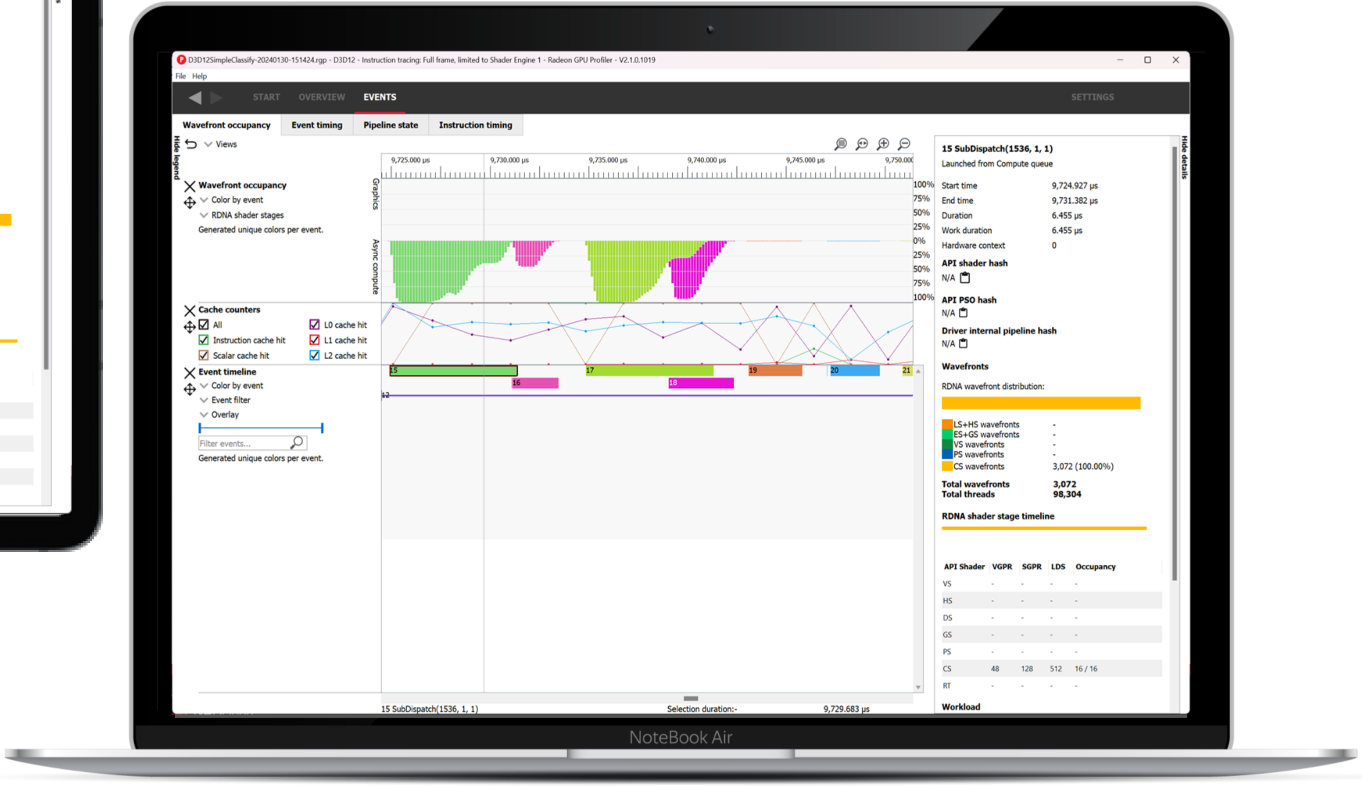
RDNA wavefront distribution:

🟧 LS+HS wavefronts	-
🟩 ES+GS wavefronts	-
🟩 VS wavefronts	-
🟦 PS wavefronts	-
🟨 CS wavefronts	3,072 (100.00%)
Total wavefronts	**3,072**
Total threads	**98,304**

RDNA shader stage timeline

API Shader	VGPR	SGPR	LDS	Occupancy
VS	-	-	-	-
HS	-	-	-	-
DS	-	-	-	-
GS	-	-	-	-
PS	-	-	-	-
CS	48	128	512	16 / 16
RT	-	-	-	-

Learn more in our
AMD Radeon™ Tools session (YouTube link)



15 SubDispatch(1536, 1, 1)   Selection duration:-   9,729.683 μs

15 SubDispatch(1536, 1, 1)

Launched from Compute queue

Start time	9,724.927 μs
End time	9,731.382 μs
Duration	6.455 μs
Work duration	6.455 μs
Hardware context	0

API shader hash

N/A

API PSO hash

N/A

Driver internal pipeline hash

N/A

Wavefronts

RDNA wavefront distribution:

🟧 LS+HS wavefronts	-
🟩 ES+GS wavefronts	-
🟩 VS wavefronts	-
🟦 PS wavefronts	-
🟨 CS wavefronts	3,072 (100.00%)
Total wavefronts	**3,072**
Total threads	**98,304**

RDNA shader stage timeline

API Shader	VGPR	SGPR	LDS	Occupancy
VS	-	-	-	-
HS	-	-	-	-
DS	-	-	-	-
GS	-	-	-	-
PS	-	-	-	-
CS	48	128	512	16 / 16
RT	-	-	-	-

Workload

WHAT ABOUT VULKAN? 🌋

Work graphs are also coming to Vulkan®

Currently, AMDX

(AMD only, experimental)

As usual…

- Want to match D3D with a EXT/KHR extension
- We plan to release updates to the AMDX in tandem with new features in D3D (like draw calls)

WORK GRAPHS SUMMARY

1

GPU managed producer/consumer **networks**

- ✓ with expansion/reduction
- ✓ with recursion

2

GPU managed **memory** – can never run out of memory

3

Guaranteed **forward progress**: No deadlocks, no hangs, by construction

Available now!

https://gpuopen.com/microsoft-work-graphs-1-0-now-available/

THANKS! NOW, GO TRY IT OUT!

Head over to https://gpuopen.com/microsoft-work-graphs-1-0-now-available/

Big thanks also go out to:

- Amar Patel & Shawn from Microsoft
- the fine folks at the university of Coburg (Bastian Kuth, Quirin Meyer, Carsten Faber),
- the whole team at AMD, specifically Rob Martin, Max Oberberger, Niels Fröhling, Pirmin Pfeiffer, Dominik Baumeister, Timothy McQuaig, Jason Stewart, and many more

and everyone else who made this a reality!

DISCLAIMER

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

“Mesh nodes: Performance” - Testing by AMD as of March 15, 2024, on the AMD Radeon RX 7900 XTX using AMD Software: Adrenalin Edition 31.0.24014.1002 pre-release driver, using the ExecuteIndirect command and Work Graphs with the mesh nodes extension to dispatch scene information to Microsoft® DirectX® 12, on a test system configured with an AMD Ryzen™ 7 5800X CPU, 32GB DDR4 RAM, Gigabyte X570 AORUS ELITE WIFI motherboard, and Windows 11 Pro 2023 Update, using the AMD procedural content Work Graphs demo with the overview, meadow, bridge, wall, and market scene views. System manufacturers may vary configurations, yielding different results. RS-640.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

© 2024 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, Radeon, RDNA, Ryzen, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective owners. DirectX is either a registered trademark or trademark of Microsoft Corporation in the US and/or other countries. Vulkan and the Vulkan logo are registered trademarks of the Khronos Group Inc. Xbox is a registered trademark of Microsoft Corporation in the US and/or Other countries.