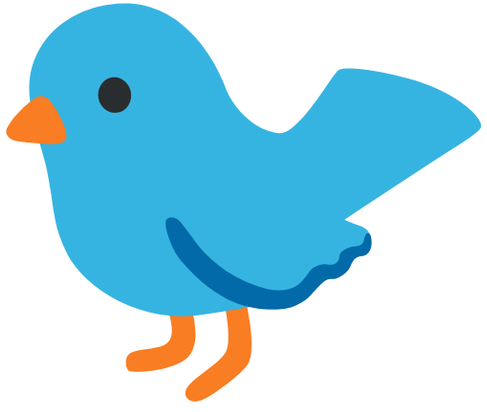
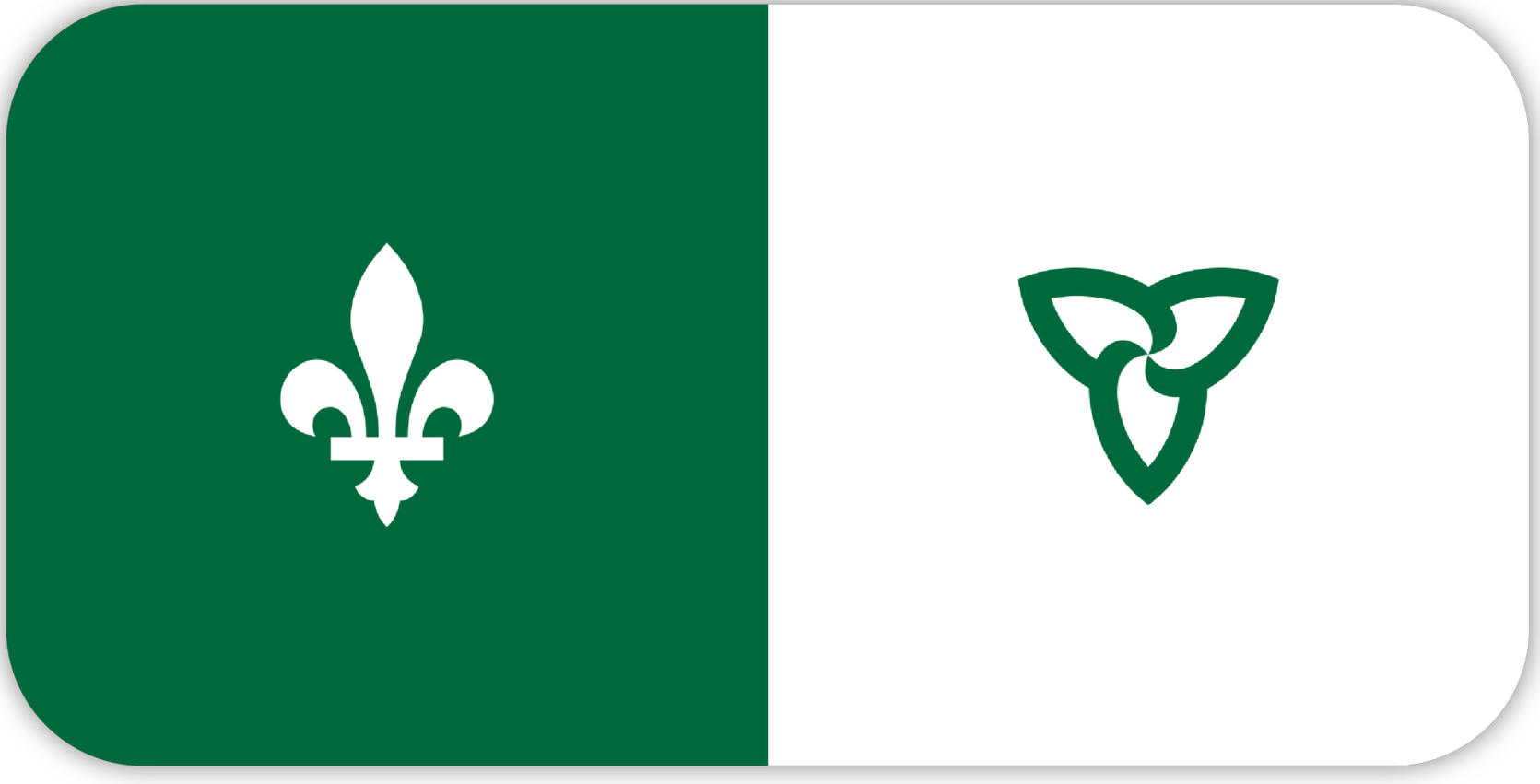
Le compilateur Jay

L’assignation unique statique pour les GPU Intel

Crédit image : Rob Hanson de Welland, Ontario, Canada, CC BY 2.0, via Wikimedia Commons

 Pourquoi?





The Jay compiler

Static single assignment for Intel GPUs

Image credit: Rob Hanson from Welland, Ontario, Canada, CC BY 2.0, via Wikimedia Commons

 Why?

🐦 Compile-time

“brw is so slow.”

–Everybird

Compile-time

“Windows is slower!”

Compile-time

"Windows is slower!"

–You (in 30 minutes from now)

🐦 Compile-time

- Graph colouring is ***slow***
- Linear scan hurts code quality
- Tree scan (SSA) gets great code in linear-time

🐦 As seen on TV

- ACO
- ir3
- AGX
- NAK
- [Your compiler here]

🐦 The background

- **Register pressure**: max live variables
- **Register demand**: # of registers used

🐦 The background

- **Register pressure**: max live variables
- **Register demand**: # of registers used
- **In SSA**: demand equals pressure!

🐦 Applying the magic

- Calculate demand *before* register allocation!

🐦 Applying the magic

- Calculate demand *before* register allocation!
- Decouple spilling & register assignment

🐦 Applying the magic

- Calculate demand *before* register allocation!
- Decouple spilling & register assignment
- Scheduler that never spills

 So...

Let’s do tree scan on Intel!

Image credit: Andy Morffew from Itchen Abbas, Hampshire, UK, CC BY 2.0 via Wikimedia Commons Rob Hanson from Welland, Ontario, Canada

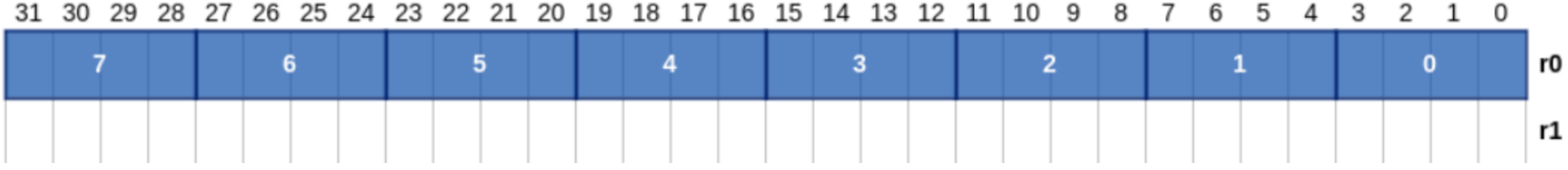
The hardware

Intro to GenISA

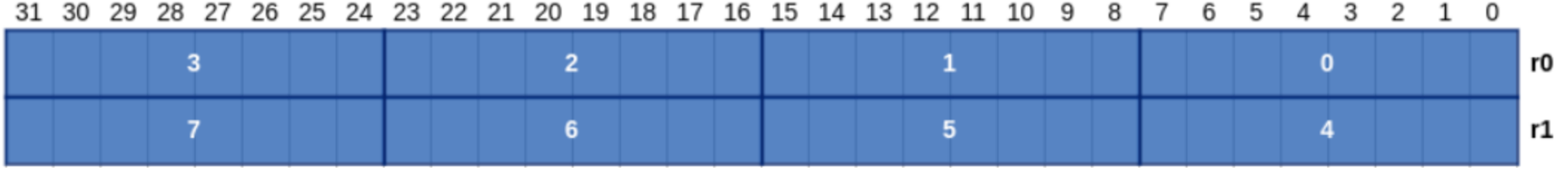
```
(f0) add (32) r0<1>:d r1<1,1,0>:d r2.6<0,1,0>:d
```

- Explicit SIMD
- 256/512-bit registers, up to SIMD32
- Destination strides
- Source regions
- Predication
- Flexible in theory, restricted in practice

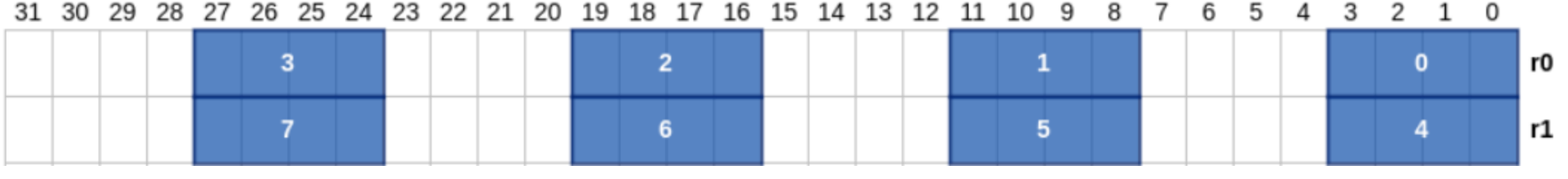
Strides



32-bit



64-bit



32-bit value with 64-bit stride

🐦 The GenISA problem

- **Cross-channel interference**
- Lane 0 writes clobbers lane 1 data
- …if control flow diverges
- …and strides differ

This code is wrong!

```
if (lane % 2) {
    r0    = load.i32 a[lane]
} else {
    r0:r1 = load.i64 b[lane]
    r0    = iadd.i32 r0, r1
}
```

r0 write in the “else” clobbers r0 in the “if”!

🐦 Now what?

Tree scan won’t work.

The solution

🐦 Make-believe!

Pretend you are a GPU with a sane design.

Pretend you are a GPU with a sane design.

Copy ACO/NAK.

Pretend you are a GPU with a sane design.

Copy ACO/NAK.

Make no mistakes.

~~Pretend you are a GPU with a sane design.~~

~~Copy ACO/NAK.~~

~~Make no mistakes.~~

This slide is a joke.

~~Pretend you are a GPU with a sane design.~~

~~Copy ACO/NAK.~~

~~Make no mistakes.~~

This slide is a joke.

I only use dark ***MGC***.

Normalize the ISA!

Instead of...

```
(f0) add (32) r0<1>:d r1<1,1,0>:d r2.6<0,1,0>:d
```

Let’s ignore the crazy...

```
(32&f0) add.s32 r0, r1, u2.6
```

🐦 Defer everything!

- Fully SSA
- Per-lane GPRs
- Uniform registers (UGPRs a.k.a SGPRs)
- Flags that spill to UGPRs
- No strides/regions

Partitions

GRF	GPR	Stride
12…35	0…11	32-bit
36…55	12…21	64-bit
56…63	22…25	16-bit

GRF	UGPR
0…11	0…191

GRF	FLAG
0…7	0…3

🐦 Partitions

- Partition the register file by stride
- Cross-channel interference solved *implicitly*

🐦 Then what?

- Compile to our fantasy target
- Apply the partition to emit machine code
- Blocks have an implicit stride/region
- Easy as π

UGPR interference

This is valid:

```
r0 = load A

if (lane % 2) {
    r0 = load B
    store(r0)
} else {
    store(r0)
}
```

This is not:

```
u0 = load A

if (lane % 2) {
    u0 = load B
    store(u0)
} else {
    store(u0)
}
```

🐦 Two control flow graphs

- **Physical** for UGPRs, **logical** for GPRs
- No unnecessary interference for GPRs
- **Critical edges** in the physical CFG

Register assignment

- Colombet et al
- Got RA constraints? Repair!
- Local, not global
- Tolerates critical edges

🐦 Regioning restrictions

- Regioning restrictions → stride constraints
- Partition maps registers → strides
- Restrictions are local RA constraints
- No spilling!

🐦 Correctness is easy!

- Vectors, strides, alignment
- Assign *all* operands for every instruction
- …in descending order of size
- …guaranteeing correctness with vectors

Performance is not!

- Cost model: how many moves does this take?
- Roundrobin heuristic
- …Better post-RA scheduling
- …Less fragmentation for vectors
- …Cost function optimization is often O(1)

🐦 Spilling

- Braun-Hack
- Spill at definitions (less divergence!)
- Fill UGPRs conservatively
- Tolerates critical edges

Hardware support

🐦 Less hardware

- **Integrated**: Lunar Lake, Panther Lake
- **Discrete**: Battlemage

 More hardware

- **Xe2+**: supported!
- **Skylake–Xe1**: eventually™
- **Broadwell**: never, sorry

🐦 More hardware

- **Xe2+**: supported!
- **Skylake–Xe1**: eventually™
- **Broadwell**: never, sorry
- **Sandybridge**: non, jamais

🐦 More hardware

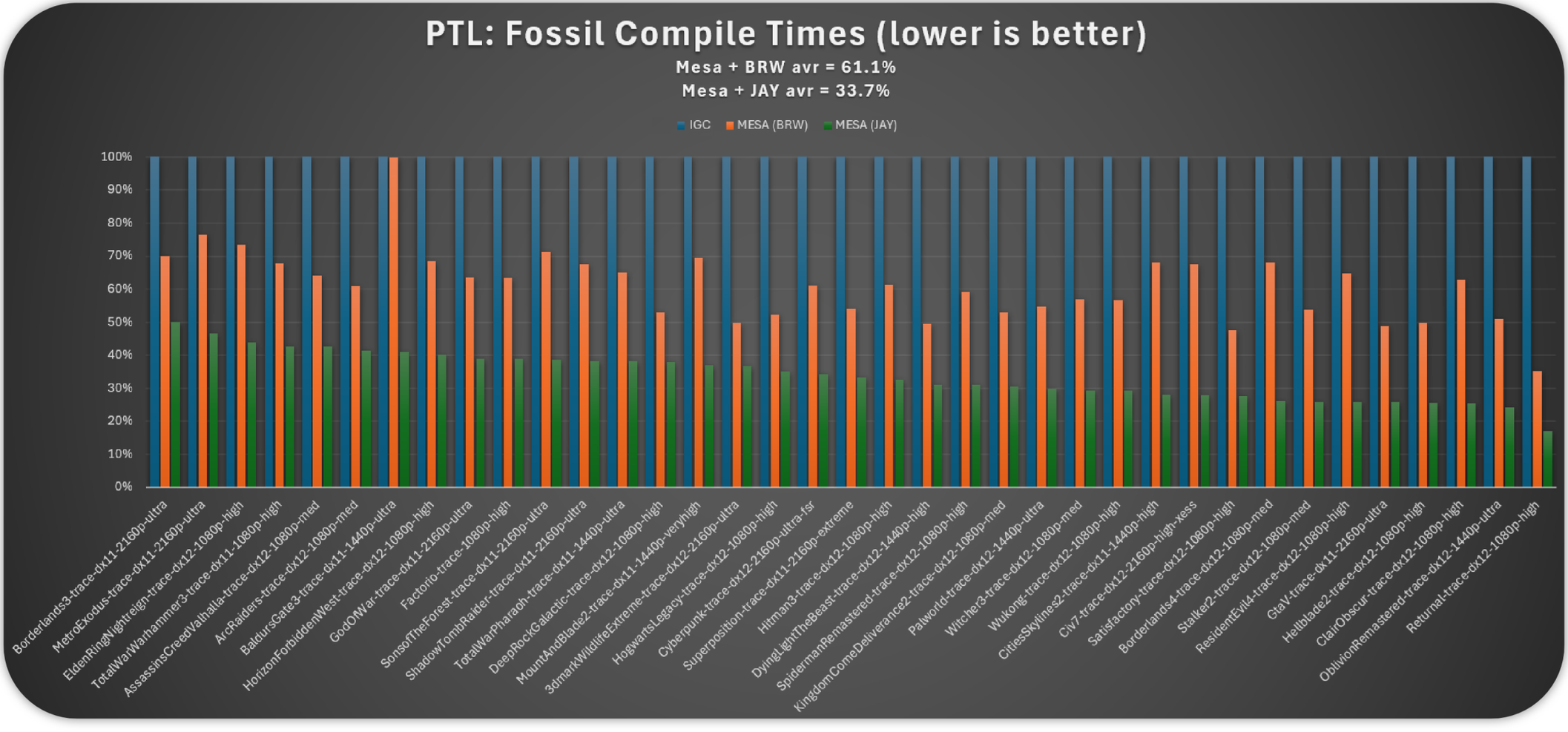
- **Xe2+**: supported!
- **Skylake–Xe1**: eventually™
- **Broadwell**: never, sorry
- **Sandybridge**: non, jamais
- **Ironlake**: ferme ta yeule

🐦 More hardware

- **Xe2+**: supported!
- **Skylake–Xe1**: eventually™
- **Broadwell**: never, sorry
- **Sandybridge**: non, jamais
- **Ironlake**: ferme ta yeule
- **Pentium**: câlisse

Compile-time

Compile-time



55% faster than brw! **66%** faster than Windows!

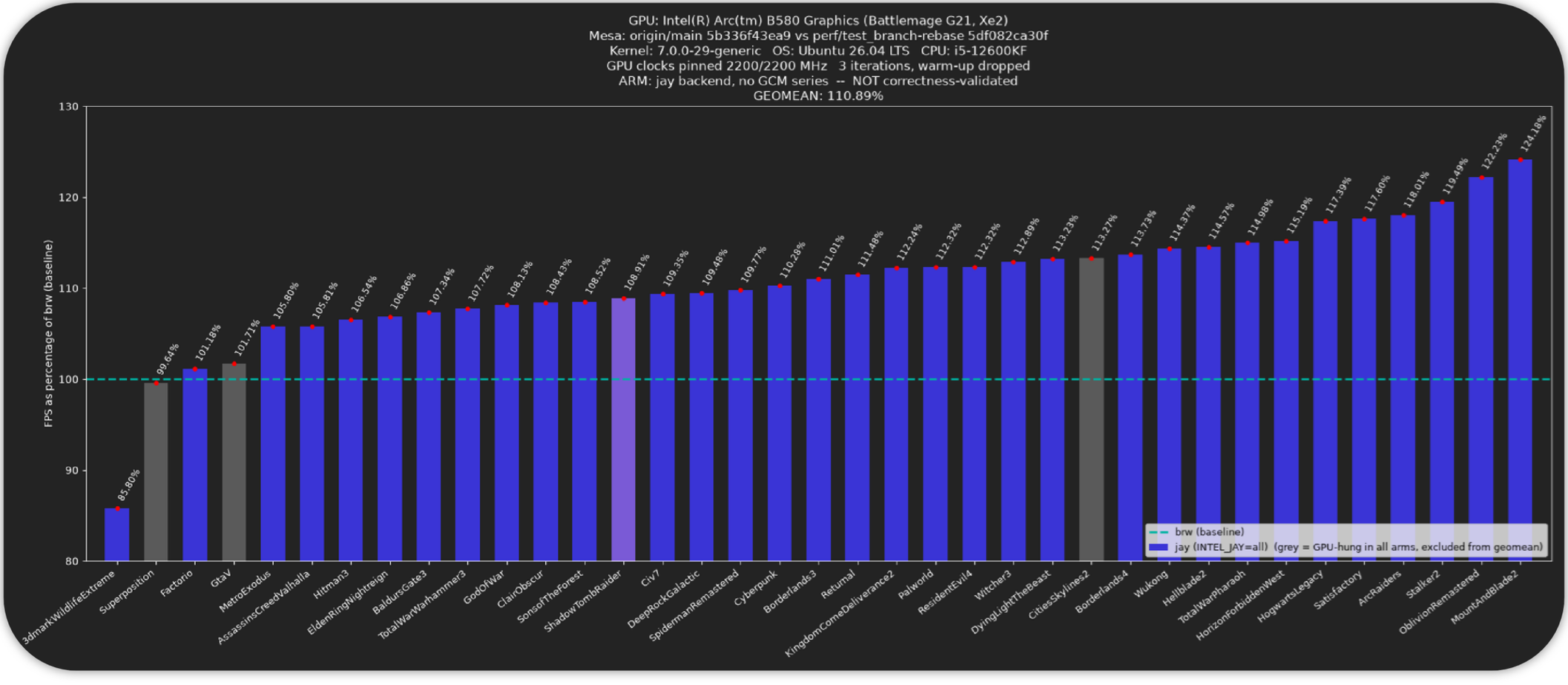
Compile-time

“Windows is slower!”

“Windows is slower!”

Game performance

MOAR FPS!



10% faster than brw!

 When?

🐦 When?

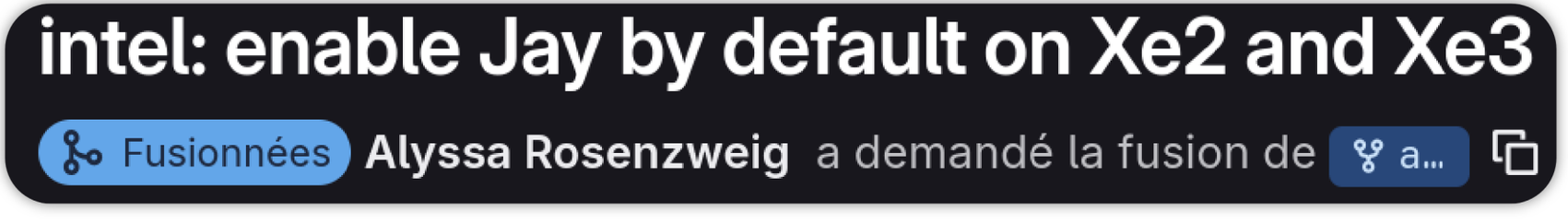
Haven’t you met me?

Obviously

Now

~~Now~~

Last Thursday



Credit reel

Authors

- Alyssa Rosenzweig
- Kenneth Graunke
- Calder Young
- Caio Oliveira
- Sagar Ghuge
- Lionel Landwerlin
- Adrian Baker

🐦 Champions

- Alwin Mathew
- Ulisses Furquim
- Felix DeGrood

Academics

- Benoit Boissinot
- Gerhard Goos
- Matthias Braun
- Quentin Colombet
- Sebastian Hack

Pioneers

- Daniel Schürmann
- Connor Abbott
- Faith Ekstrand

Kraid

Thank you

Alyssa Rosenzweig