22. – 24.
september
2025

INT30 KONF

Hybrid Caching in .NET

Jody Donetti

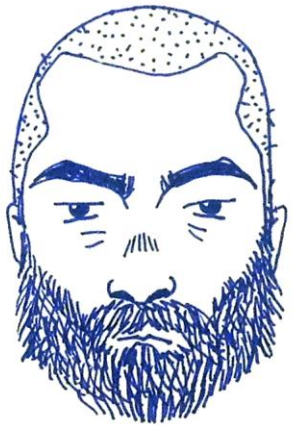
Jody Donetti

Coding + R&D

Doing stuff (mainly) on the web for around 30 years.

Dealt with most types of caches: memory, distributed, hybrid, HTTP caching, offline caching, CDNs.

Created FusionCache, a .NET hybrid caching library.



🏆 Google Open Source Award

🏆 Microsoft MVP Award

Hybrid What?

🤔 Hybrid what?

What is this “hybrid cache” thing?

And does it make sense to use one?

Well, first things first, there are 3 main types of caches:

- memory caches
- distributed caches
- hybrid/multi-level caches

Let’s take a journey together to make sense of them.

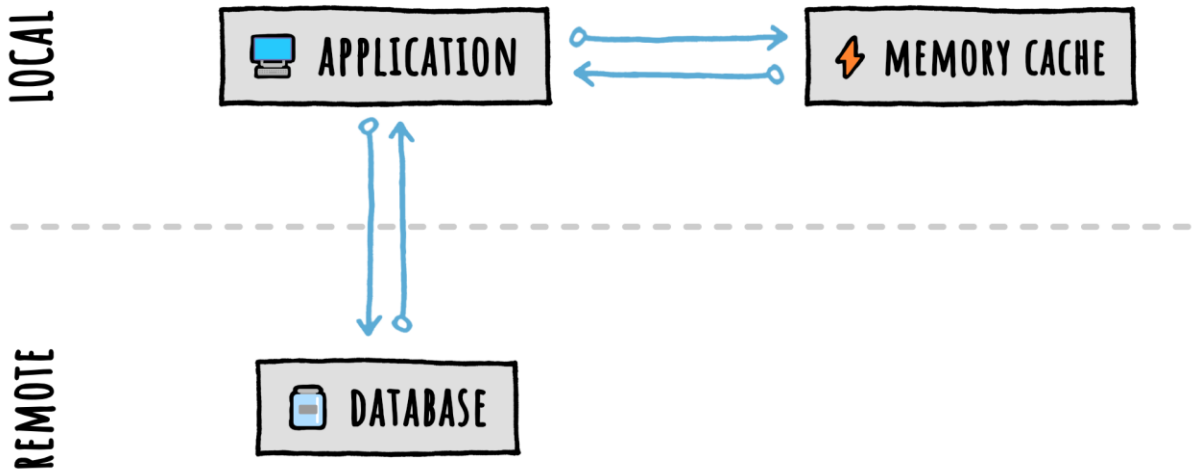
PS: I want t show a lot of cool things, so I’ll go a bit fast 😅

Memory Caches

Memory Caches

Memory caches store data in memory.
And not just “in memory”, but in the **same** memory space as the application using it.

Think of them as a dictionary + some form of eviction.



Memory Caches

We can use them like this:

```
var product = cache.GetOrCreate<Product>(
    $"product:{id}",
    // WARNING: NO STAMPEDE PROTECTION
    _ => GetProductFromDb(id),
    options
);
```

Memory Caches

Some examples of **memory caches** in .NET:

📦 **BitFaster.Caching**
github.com/bitfaster/BitFaster.Caching

📦 **FastCache**
github.com/jitbit/FastCache

📦 **fast-cache**
github.com/neon-sunset/fast-cache

📦 **LazyCache**
github.com/alastairtree/LazyCache

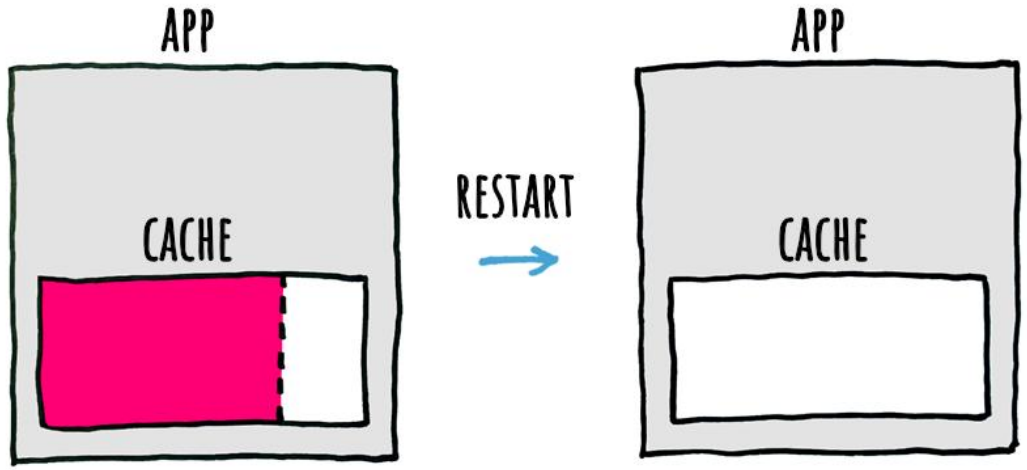
Ⓜ️ **Microsoft MemoryCache**

Cold Starts

Cold Starts

Ok, so we are using a **memory cache**, cool: but what happens when the **app restarts**?

This:



The memory cache is now **empty**.

🥶 Cold Starts

Remember: a memory cache is just an in-memory dictionary.

And it needs to be **re-populated** again from the **database**.

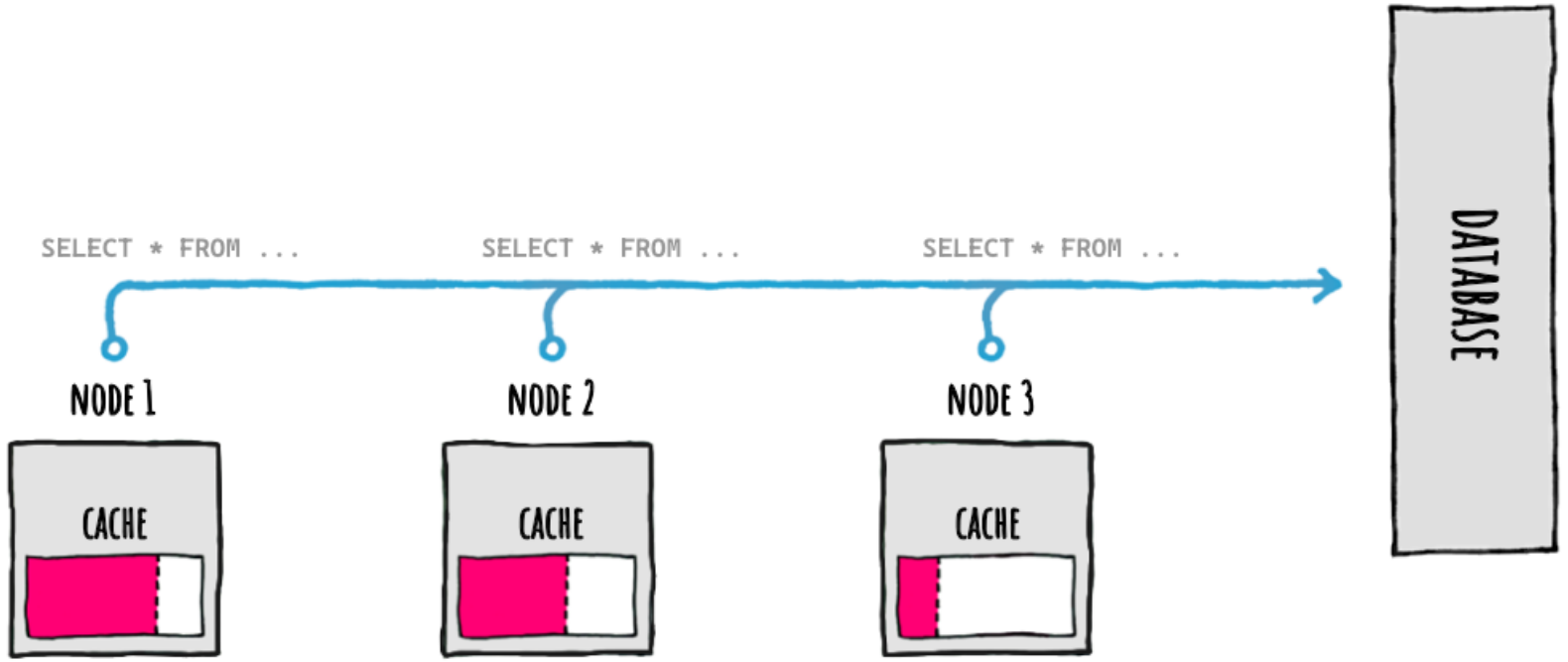
This means **more database queries**.

Ok, anything else?

Horizontal Scalability

Horizontal Scalability

What if our app is deployed on **multiple instances**, **nodes**, or **pods**?
This:



Each instance/node/pod has **its own** local cache.

🪜 Horizontal Scalability

Each one needs to be **populated** from the **database**.
All because cache data is **not shared**.

And this, again, means **more database queries**.

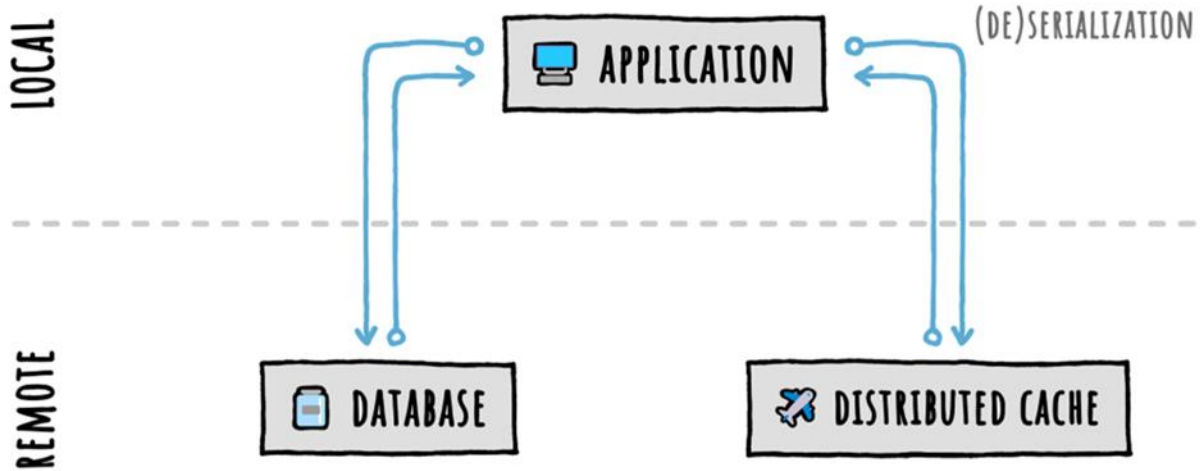
Ok, so: what can we do?

Distributed Caches

Distributed Caches

Distributed caches represent remote key-value stores (e.g.: Redis, Memcached).
Think of them as a database but **simpler**, with less features: because of that, way **faster**.

In .NET it's `IDistributedCache` and available implementations, and the talk `byte[]`.



Distributed Caches

We can use them like this:

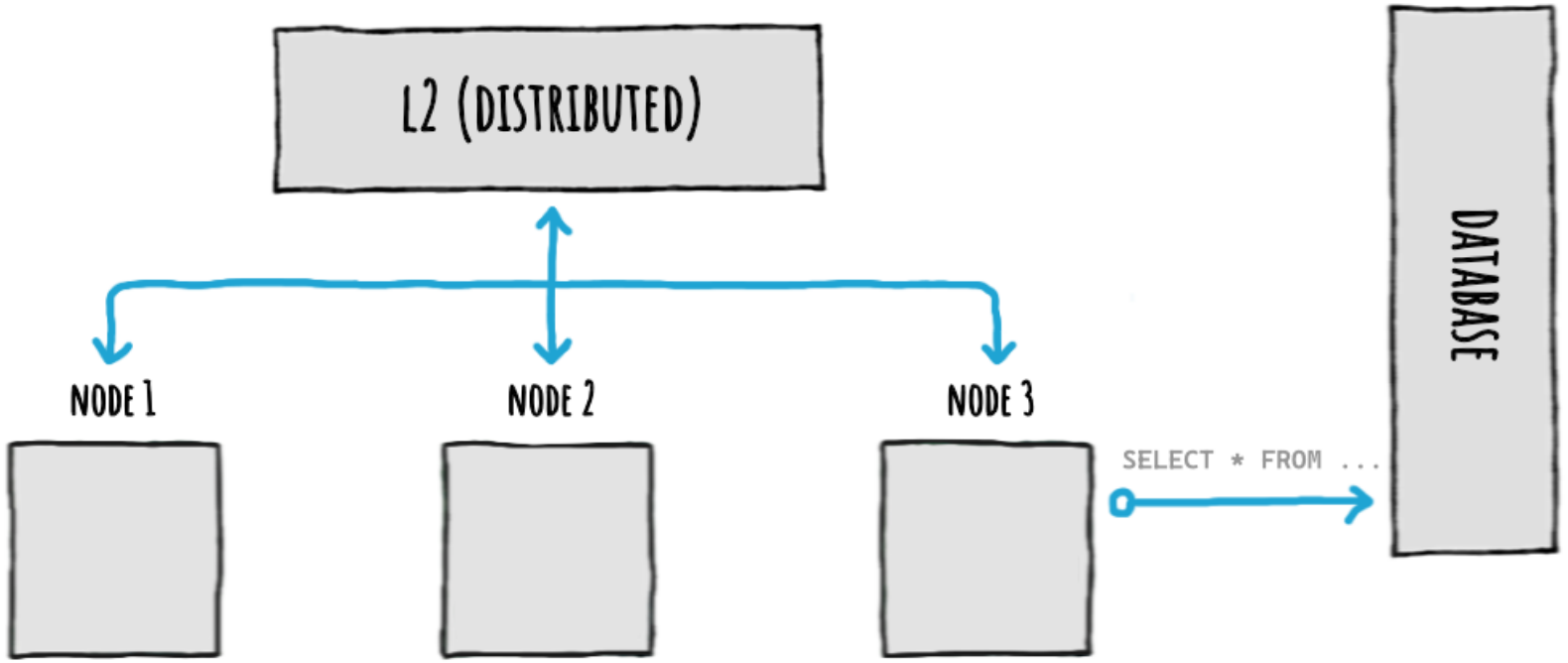
```
Product product;
var payload = distributedCache.Get($"product:{id}");
if (payload is not null)
{
    product = JsonSerializer.Deserialize<Product>(payload);
}
else
{
    // WARNING: NO STAMPEDE PROTECTION
    product = GetProductFromDb(id);
    payload = JsonSerializer.SerializeToUtf8Bytes<T?>(product);
    distributedCache.Set($"product:{id}", payload);
}
```

Distributed Caches

Since distributed caches are **remote**, cached data lives **outside** of the memory space of the app.

So it:

- can **survive** app restarts
- can be **shared** between multiple nodes



✈️ Distributed Caches

Some examples of **distributed caches** in .NET (`IDistributedCache` implementations):

📦 **EnyimMemcachedCore** (for Memcached)
github.com/cnblogs/EnyimMemcachedCore

📦 **MongoDbCache** (for MongoDB)

github.com/outmatic/MongoDbCache

📦 **NeoSmart.Caching.Sqlite** (for SQLite)
github.com/neosmart/SqliteCache

📦 **AWS.AspNetCore.DistributedCacheProvider** (for Amazon DynamoDB)
github.com/aws/aws-dotnet-distributed-cache-provider/

Ⓜ️ **Microsoft.Extensions.Caching.StackExchangeRedis** (for Redis)

✈️ Distributed Caches

Some examples of **distributed caches** in .NET (`IDistributedCache` implementations):

📦 **EnyimMemcachedCore** (for Memcached)
github.com/cnblogs/EnyimMemcachedCore

📦 **MongoDbCache** (for MongoDB)

github.com/outmatic/MongoDbCache

📦 **NeoSmart.Caching.Sqlite** (for SQLite)
github.com/neosmart/SqliteCache

📦 **AWS.AspNetCore.DistributedCacheProvider** (for Amazon DynamoDB)
github.com/aws/aws-dotnet-distributed-cache-provider/

Ⓜ️ **Microsoft.Extensions.Caching.StackExchangeRedis** (for Redis)

Distributed != Distributed

💾 Distributed != Distributed

A **distributed cache** may not be physically **“distributed”**.

Basically, `IDistributedCache` can mean “an out-of-process cache that talks `byte[]`”.

For example:

📦 **NeoSmart.Caching.Sqlite**
github.com/neosmart/SqliteCache

can be a good choice for things like a **desktop/mobile app**, to avoid cold starts.

All **without** having to manage a **separate service** like Redis, Memcached, etc.

So, just distributed?

🤔 So, just distributed?

When working **directly** with a distributed cache, data is **shared** between multiple nodes.

Nice.

But we must also consider other things:

- **code:** more code needed (e.g.: manual (de)serialization)
- **cost:** for every single request, we pay the price of network + (de)serialization
- **availability:** since it's distributed, it may not always be available or reachable
- **stampede:** no protection

Wait, what is this "stampede" thing?

Cache Stampede

Cache Stampede

Imagine this scenario.

Multiple requests arrive to our app/service, all for the **same data** (not yet in the cache), all at the **same time**.

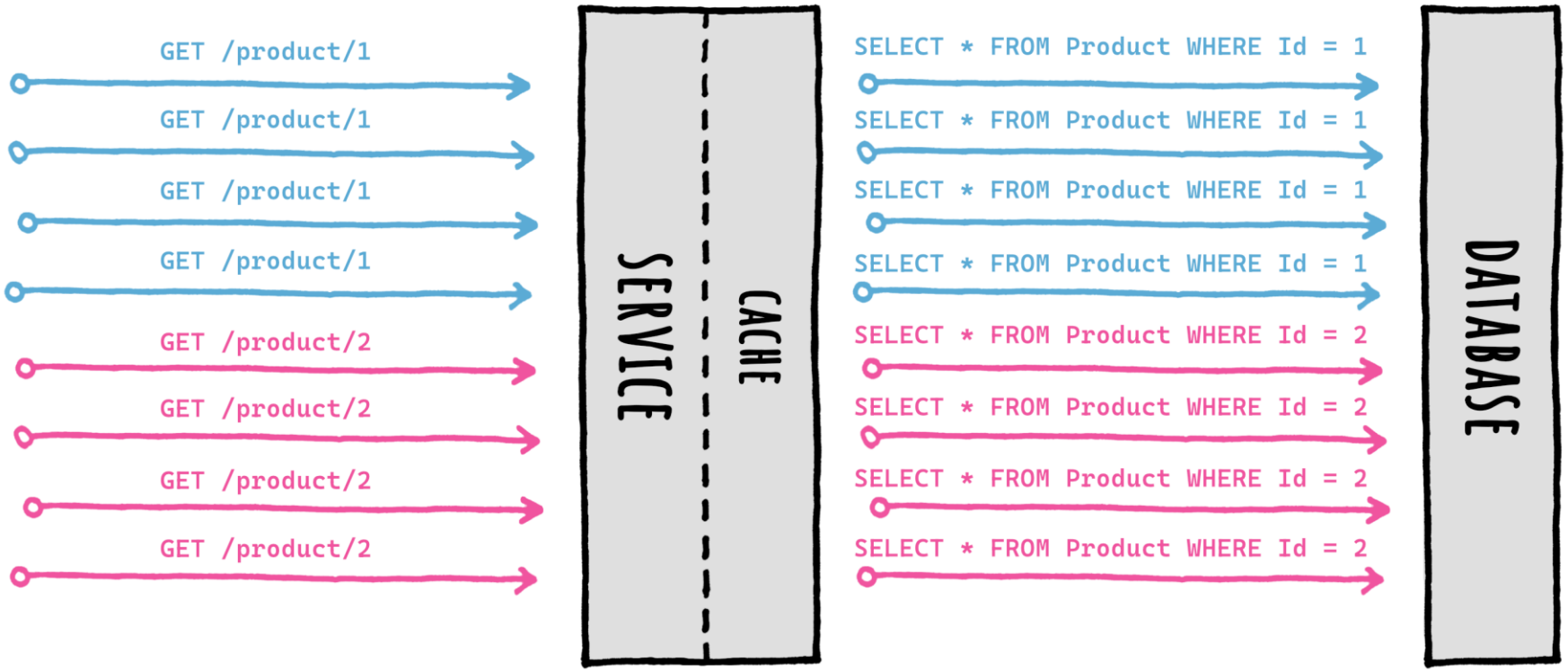
Without any special care, **each request** would:

- **cache read:** try to load the cache for the data
- **cache miss:** not find anything in the cache
- **factory:** get data from the database
- **cache write:** save the data in the cache

Ouch.

Cache Stampede

Basically, on a **cache miss**, we have this:



Cache Stampede

Now: imagine the same scenario with **100** or **1,000** or even more concurrent requests.

This would be a **huge waste** of time, resources and something that can easily tear down our database.

And maybe during peak traffic time, on a Black Friday.
Because of course, right?

Nice to meet you, **Cache Stampede**.

So, what can be done?

Cache Stampede

Some caching libraries (**not all**) have integrated Cache Stampede protection.

They coordinate:

- cache calls (get/set)
- factory execution (database query)

for the **same cache key** at the **same time**, all automatically.

But we need to give them **a chance** to protect us.

Cache Stampede (bad)

Don’t do **separate** calls:

```
// CACHE READ
var product = cache.Get<Product>($"product:{id}");

// CACHE MISS CHECK
if (product is null)
{
    // DATABASE READ
    product = GetProductFromDb(id);
    // CACHE WRITE
    cache.Set<Product>($"product:{id}", product);
}
```

Cache Stampede (good)

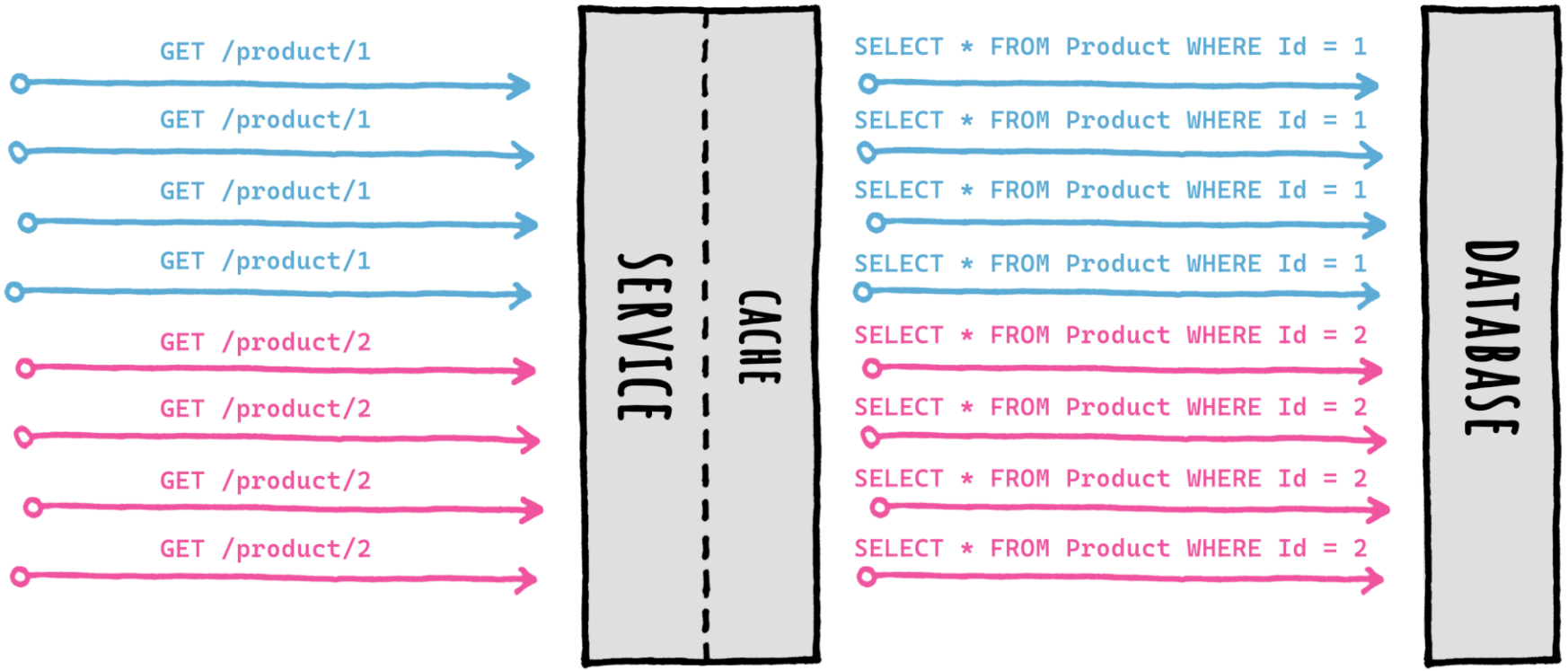
Instead do **one call**, allowing the cache to **coordinate** everything:

```
var product = cache.GetOrSet<Product>(
    $"product:{id}",
    // FACTORY (DATABASE READ)
    _ => GetProductFromDb(id)
);
```

ℹ️ **NOTE:** the method can be called `GetOrCreate()`, `GetOrAdd()`, etc... naming is hard.

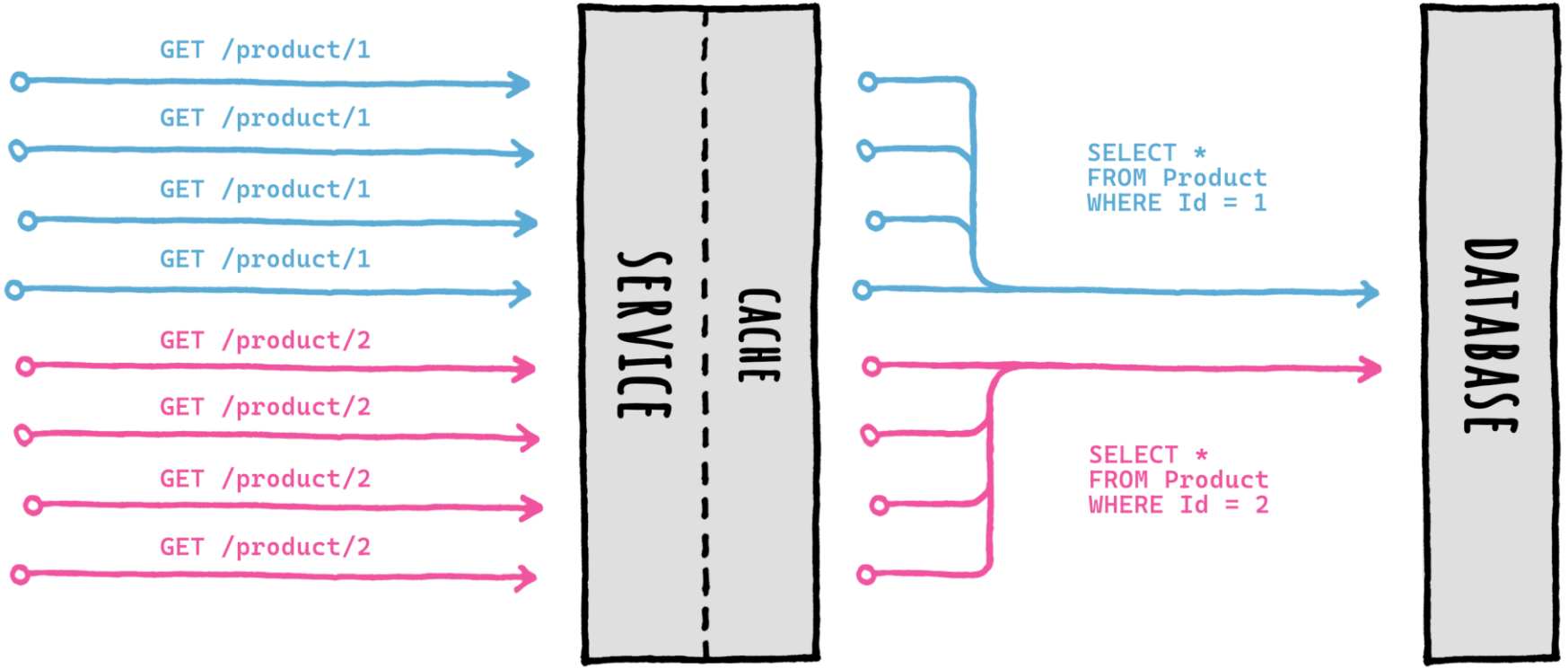
Cache Stampede

Turning this...



Cache Stampede

... into this:



🛡️ Cache Stampede

It's a form of **request coalescing**, where multiple requests are coalesced into one.

Frequently, people think that:

- **IF** there is a method like `GetOrSet(key, factory)` or similar
- **THEN** the library has stampede protection

Nope, not true:

⛔ **MemoryCache:** no protection, even with `GetOrCreate()`/`GetOrCreateAsync()`

✅ **FusionCache:** protection with `GetOrSet()`/`GetOrSetAsync()`

✅ **HybridCache:** protection with `GetOrCreateAsync()`

Also, distributed caches in general **don't protect** from cache stampede.

So, again: just distributed?

🤔 So, again: just distributed?

Again, when working **directly** with a **distributed** cache:

- more code
- network + serialization cost
- not always available
- no stampede protection

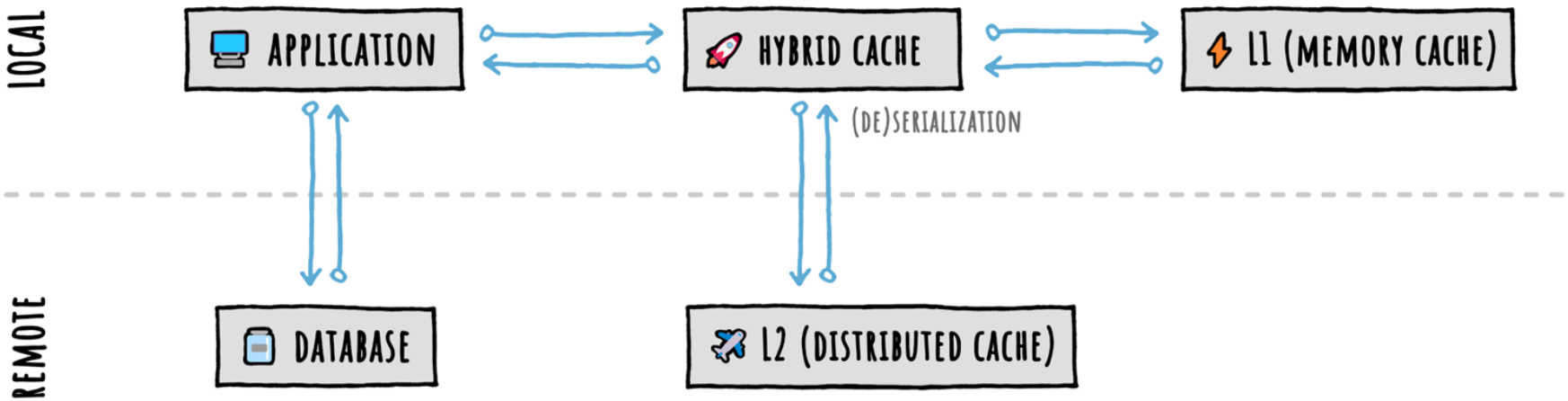
And this is where **hybrid caches** come into the picture.

Hybrid Caches

🚀 Hybrid Caches

Hybrid caches are the most advanced caches.
They combine the best of both worlds, together: memory (L1) + distributed (L2).

The “dance” between the 2 levels is taken care of, automatically.



🚀 Hybrid Caches

We can use them like this:

```
var product = cache.GetOrSet<Product>(
    $"product:{id}",
    _ => GetProductFromDb(id),
    options
);
```

🚀 Hybrid/Multi-Level Caches

Some examples of **hybrid/multi-level** caches in .NET:

📦 **CacheTower (multi-level)**
github.com/TurnerSoftware/CacheTower

📦 **CacheManager (multi-level)**
github.com/MichaCo/CacheManager

📦 **EasyCaching (multi-level)**
github.com/dotnetcore/EasyCaching

📦 **FusionCache (hybrid)**
github.com/ZiggyCreatures/FusionCache

Ⓜ️ **Microsoft HybridCache (hybrid)**

Hybrid VS Multi-Level

🚀 Hybrid VS Multi-Level

I am frequently using both «multi-level» and «hybrid» as if they are interchangeable.

They are **similar**, but **different**.

In particular:

- **multi-level:** any number of levels, each of any type
- **hybrid:** L1 (memory) or L1+L2 (memory + distributed)

🚀 Hybrid VS Multi-Level

As developers, we can think of them like this (no real code):

```
class MultiLevelCache
{
    List<ICacheLevel> Levels { get; }
}
```

```
class HybridCache
{
    IMemoryCache L1 { get; }
    IDistributedCache? L2 { get; }
}
```

🚀 Hybrid VS Multi-Level

Although hybrid caches may look more «limited» they are more opinionated and, in fact, more powerful.

Here's why:

- limitation is **pragmatic**, with no real-world impacts
- that ensures **stronger foundations** to build upon
- allow a **richer design** with **more advanced features**
- all while giving more **granular** control (e.g.: skip L1/L2 per-call)
- in general, they **just work**, even in complex scenarios

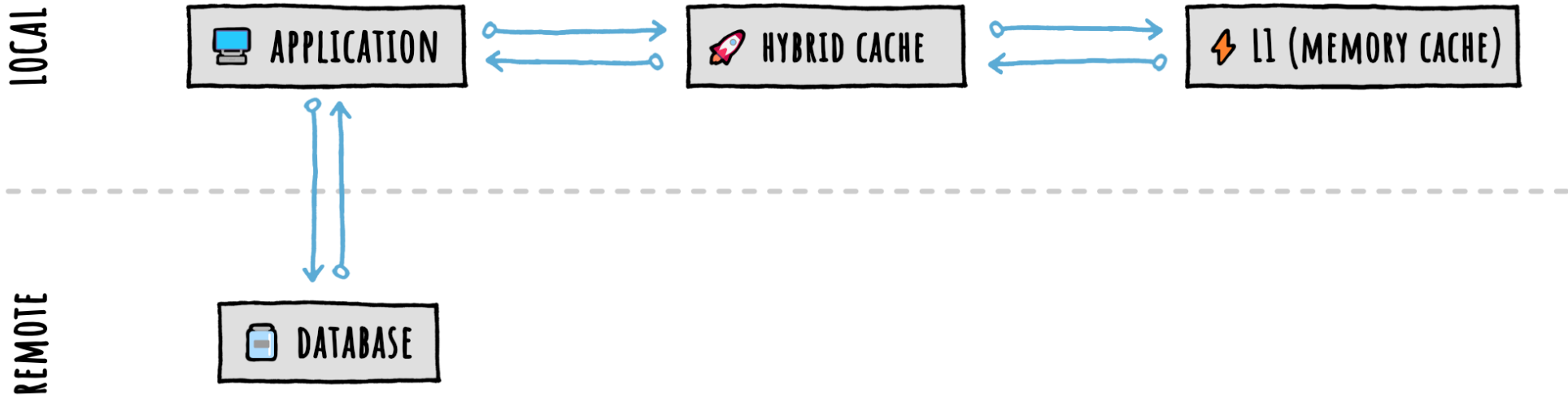
All in all, they are (imho) the right **balance**.

Hybrid != L1+L2

🚀 Hybrid != L1+L2

By using a hybrid cache you are **not forced** to use multiple levels (L1+L2).

You can use a hybrid cache with **only L1**:



Ok but... why?

🚀 Hybrid != L1+L2

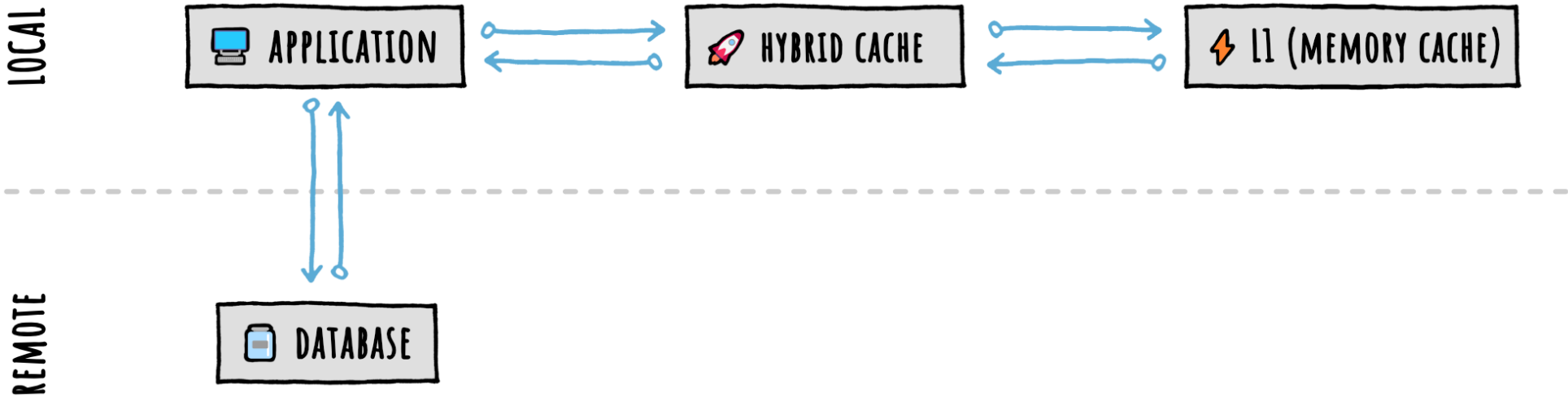
Hybrid caches are **higher level** so in general we can expect things like **more features**, **full observability**, etc.

Btw it depends on the specific library, so always check it.

But most importantly: we can **transparently** go multi-level, with **no changes** in our code.

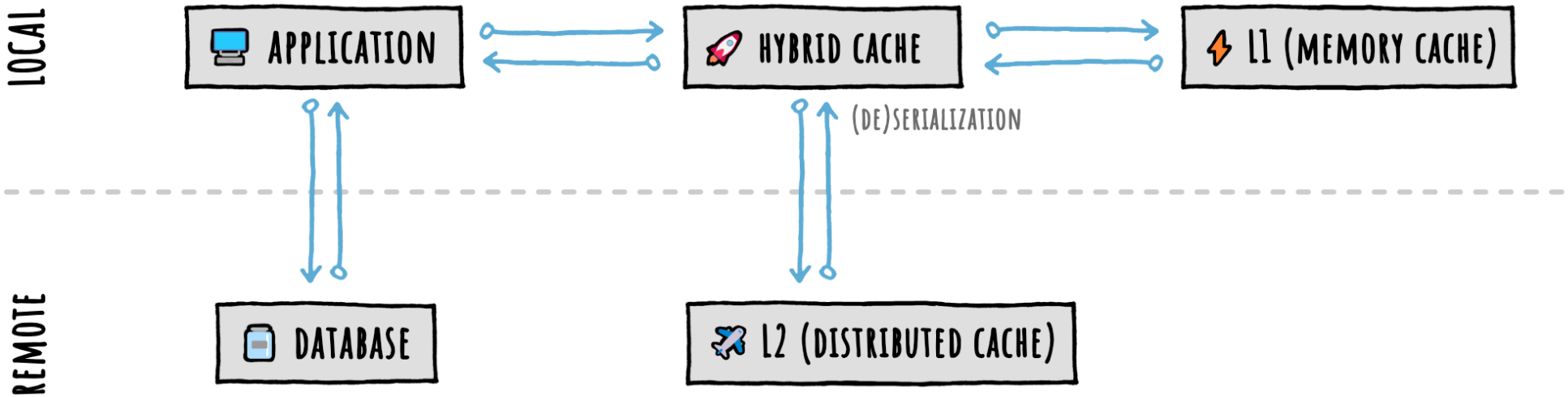
🚀 Hybrid != L1+L2

Meaning: it doesn't matter if we want **only L1**...



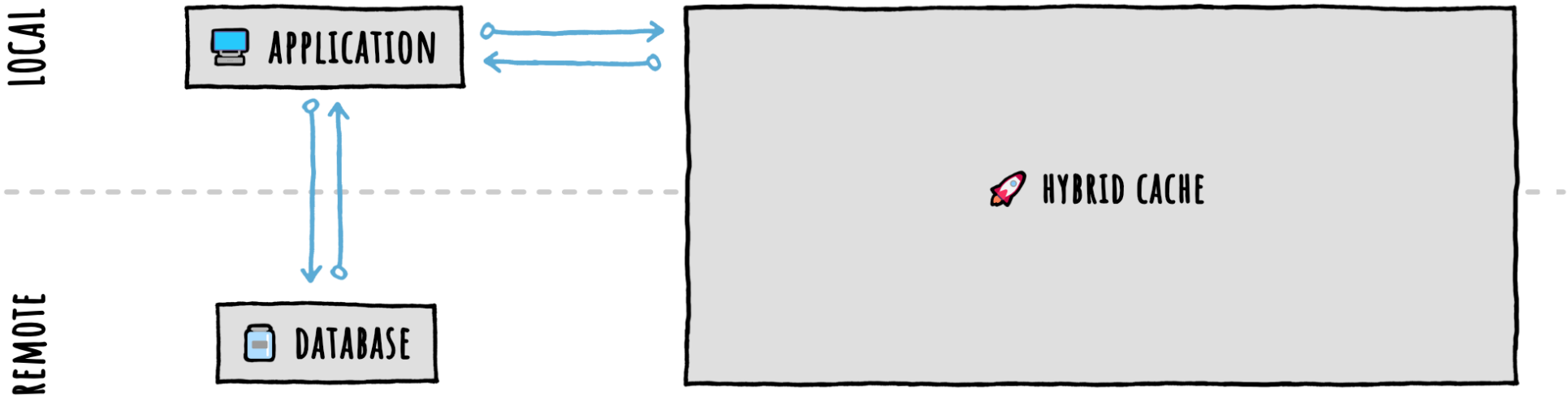
🚀 Hybrid != L1+L2

... or **L1+L2** because, either way...



🚀 Hybrid != L1+L2

... we can just **transparently** code against a single, **unified** API:



🚀 Hybrid != L1+L2

Basically, with **L1 only**:

```
var product = cache.GetOrSet<Product>(
    $"product:{id}",
    _ => GetProductFromDb(id),
    options
);
```

🚀 Hybrid != L1+L2

Or with **L1+L2**, with L2 on **Redis** + **Protobuf** serialization:

```
var product = cache.GetOrSet<Product>(
    $"product:{id}",
    _ => GetProductFromDb(id),
    options
);
```

🚀 Hybrid != L1+L2

Or with **L1+L2** again, but with L2 on **Memcached** + **JSON** serialization:

```
var product = cache.GetOrSet<Product>(
    $"product:{id}",
    _ => GetProductFromDb(id),
    options
);
```

🚀 Hybrid != L1+L2

The calling code remains **always the same**.

No need to change it everywhere, only 1 line during **setup**.

Example scenarios:

- **L1** for local dev ➡️ **L1+L2** in staging/production
- **L1** initially (not many users) ➡️ **L1+L2** when success (need to scale out)

A hybrid cache != HybridCache

🚀 A hybrid cache != HybridCache

Oh, one more thing.

When we say **“a memory cache”** we refer to the **general cache type**.

We don’t necessarily mean `MemoryCache` by Microsoft, right?

In the same way, when we say **“a hybrid cache”** we refer to the **general cache type**.

We don’t necessarily mean `HybridCache` by Microsoft (2025).

For example, `FusionCache` (2020) is also “a hybrid cache”.

Caching Libraries

📦 Caching Libraries

In this talk we'll focus on these 2:

FusionCache

HybridCache

FusionCache

FusionCache

Free + OSS

Easy to use, fast and robust hybrid cache with advanced resiliency features.

github.com/ZiggyCreatures/FusionCache

- ⬇️ 25M downloads
- ⭐ 3.2K stars
- 📜 MIT license
- 💰 Free

FusionCache

Having worked with most types of caches I faced a lot of the **real-world issues** when dealing with caching.

And saw how to **prevent** them, when possible, and how to **mitigate/solve** them, when they are inevitable.

The **OSS community** always gave me a lot: sometimes I contributed back a bit (patches, bug fixes, minor things) but never in a big way.

So I decided to do more, and in 2020 **FusionCache** was born.

FusionCache

The design:

- **L1:** uses `IMemoryCache`
- **L2:** uses `IDistributedCache`
- unified serialization interface for `byte[]` (for L2)

Any existing `IDistributedCache` implementation works: Redis, Memcached, SQLite, MongoDB, etc (there's a full list in the online docs).

It transparently handles:

- one or two **levels**
- one or more **instances/nodes**

without changing a single line of consuming code.

FusionCache

How to **register** it (after installing the package):

```
services.AddFusionCache();
```

ℹ️ **NOTE:** yes, you can also do just `new FusionCache()`

FusionCache

And you can **configure** a lot more, thanks to a **fluent builder**:

```
services.AddFusionCache()
    .WithOptions(...)
    .WithDistributedCache(...)
    .WithSerializer(...)
    .WithBackplane(...);
```

FusionCache

How to **use** it:

```
public class ProductController : Controller
{
    IFusionCache _cache;

    public ProductController(IFusionCache cache)
    {
        _cache = cache;
    }

    [HttpGet("{id}")]
    public ActionResult<Product> Get(int id)
    {
        return _cache.GetOrSet(
            $"product:{id}",
            _ => GetProduct(id),
            options => options.SetDuration(TimeSpan.FromMinutes(5))
        );
    }
}
```

FusionCache

How to **use** it:

```
public class ProductController : Controller
{
    IFusionCache _cache;

    public ProductController(IFusionCache cache)
    {
        _cache = cache;
    }

    [HttpGet("{id}")]
    public ActionResult<Product> Get(int id)
    {
        return _cache.GetOrSet(
            $"product:{id}",
            _ => GetProduct(id),
            options => options.SetDuration(TimeSpan.FromMinutes(5))
        );
    }
}
```

FusionCache

How to **use** it:

```
public class ProductController : Controller
{
    IFusionCache _cache;

    public ProductController(IFusionCache cache)
    {
        _cache = cache;
    }

    [HttpGet("{id}")]
    public ActionResult<Product> Get(int id)
    {
        return _cache.GetOrSet(
            $"product:{id}",
            _ => GetProduct(id),
            options => options.SetDuration(TimeSpan.FromMinutes(5))
        );
    }
}
```

FusionCache

FusionCache has a lot of **features** to deal with real-world **needs** and **issues**.

Time is a tyrant, so we cannot go over all of them here, but let's take a very **quick** look at some of them.

If they sound interesting, you can take a deeper look on the **online docs**.

I **marked** the ones to start with, that will give you **immediate results**.

FusionCache

Some features:

- Cache Stampede protection
- **Fail-Safe** (help with database failures)
- **Eager Refresh** (help with database slowdowns)
- **Factory Timeouts** (help with database slowdowns)
- Backplane (instant notifications on multiple instances/nodes)
- **Named Caches** (like HTTP Named Clients, but for caches)
- **Tagging** (this is insanely powerful)
- Auto-Recovery (self healing for the distributed parts)
- Observability with native OTEL support (logs, traces, metrics)

FusionCache

Also:

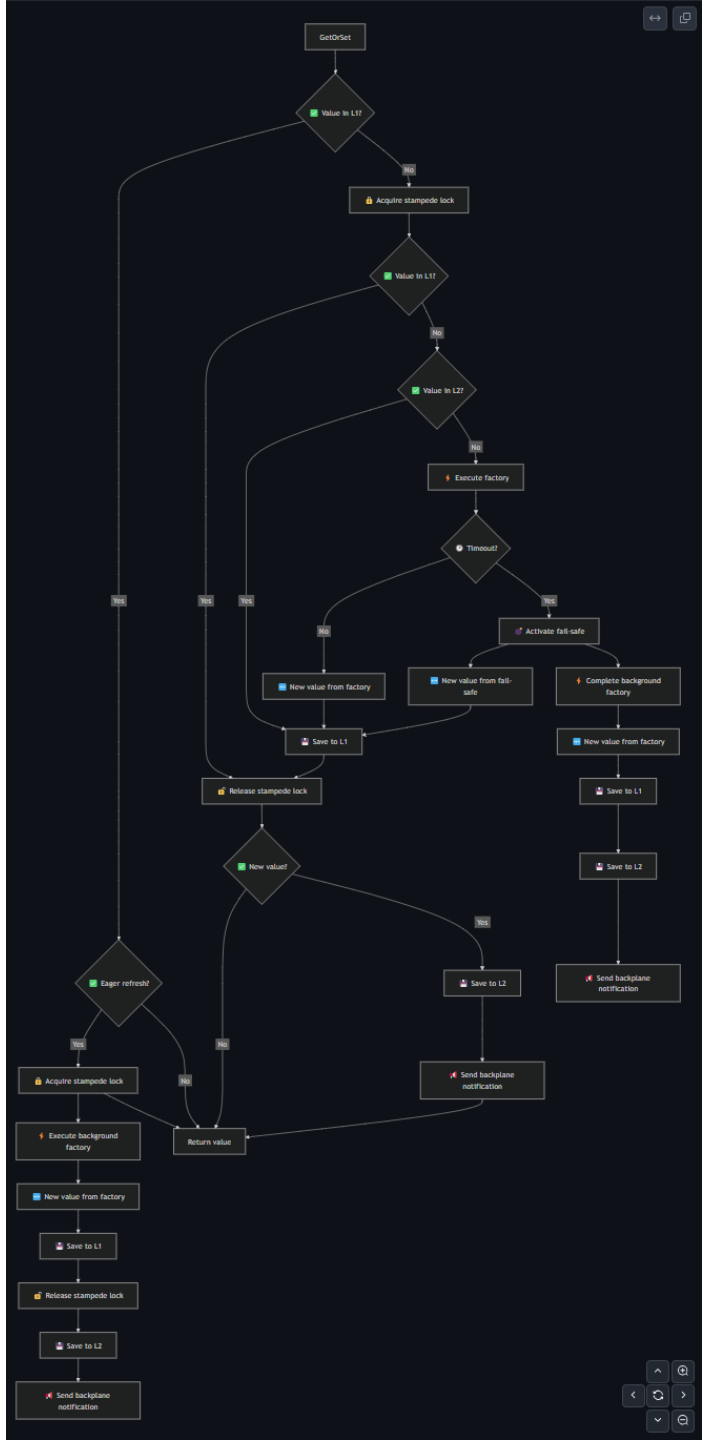
- targets .NET Standard 2.0 (runs **everywhere**, old + new)
- API is fully **sync** + **async** (no sync-over-async)
- rich **options**, both global + entry + `DefaultEntryOptions` + inheritance

FusionCache

A ton of docs, both **inline** (IntelliSense) and **online**:

- getting started
- every feature, with design rationale
- code samples
- diagrams
- step by step (bring some coffee)

Yeah, I care a lot about docs



FusionCache

Is somebody using it?

A ton of projects, both private and OSS.

A lot of companies, from small to pretty big ones like **Have I Been Pwned**, **Dometrain** and even **Microsoft** itself.

And about Microsoft: FusionCache is the caching engine in **Data API Builder** (DAB).

FusionCache

FusionCache changing license? No. **Ne.** Nada. Nein. Nope.

💰 Support

Nothing to do here.

After years of using a lot of open source stuff for free, this is just me trying to give something back to the community.

Will FusionCache one day switch to a commercial model? Nope, not gonna happen.

Mind you: nothing against other projects making the switch, if done in a proper way, but no thanks not interested. And FWIW I don't even accept donations, which are btw a great thing: that should tell you how much I'm into this for the money.

Again, this is me trying to give something back to the community.

If you really want to talk about money, please consider making 💗 **a donation to a good cause** of your choosing, and let me know about that.

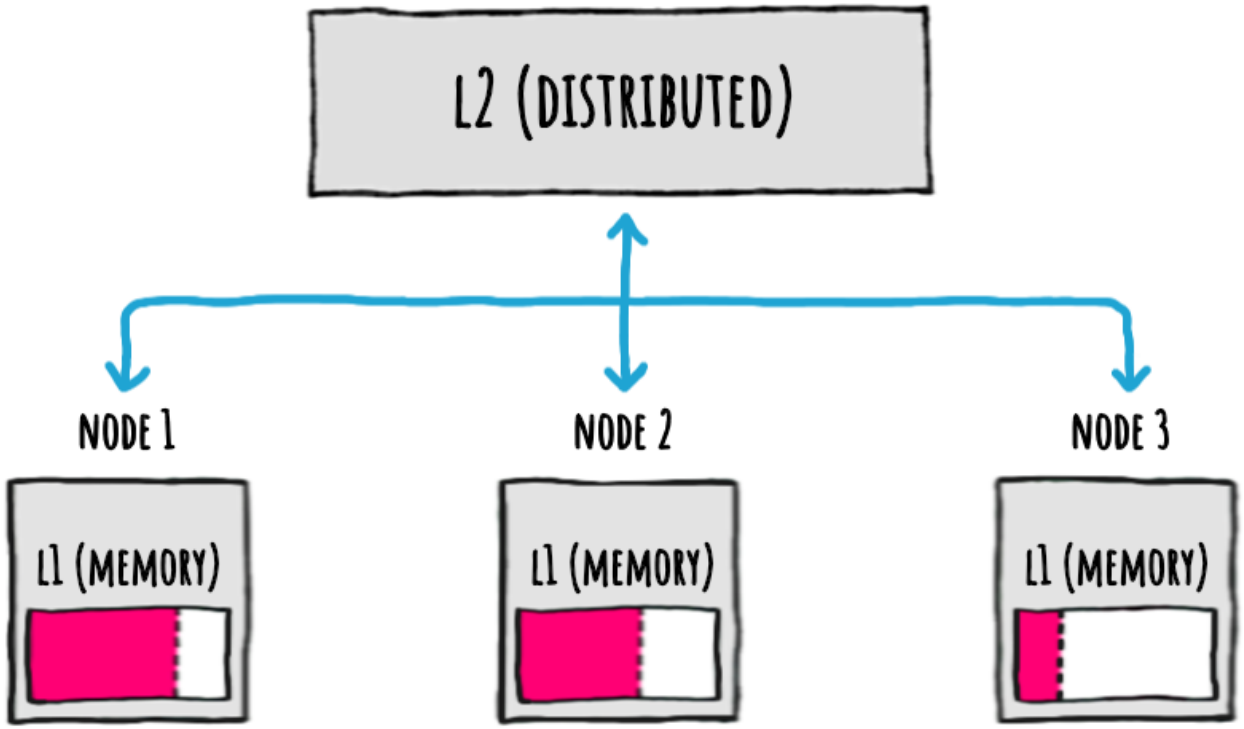
I'm just **giving back** a little bit of what I **already received**.

Cache Coherence

😨 Cache Coherence

Say we are in an **L1+L2** setup on a **multi-node** environment.

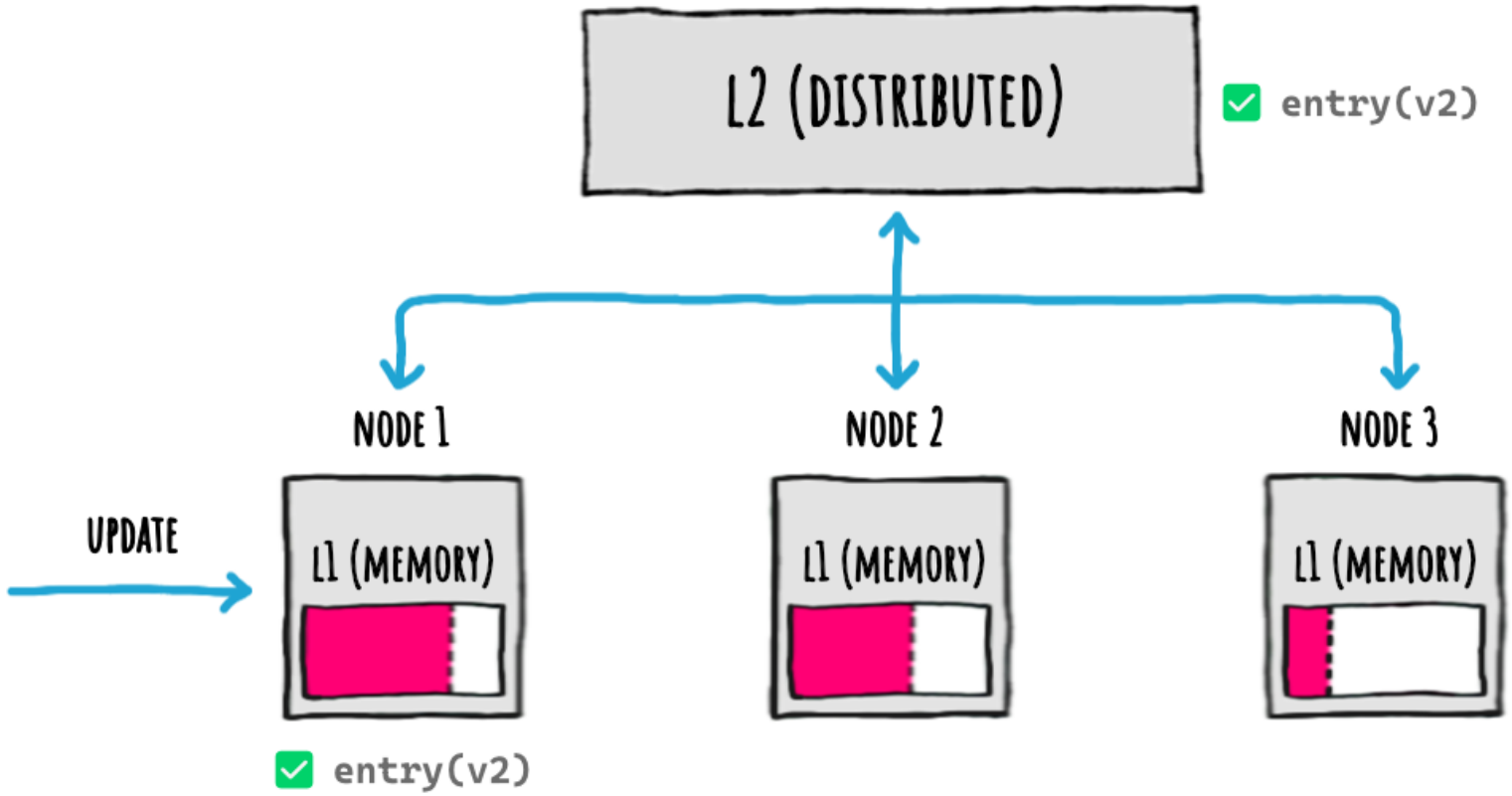
Something like this:



😨 Cache Coherence

What happens when one node does an **update** to a cache entry?

The hybrid cache writes on both **L1+L2**, but... only on the L1 **where** the **update happened**.



😨 Cache Coherence

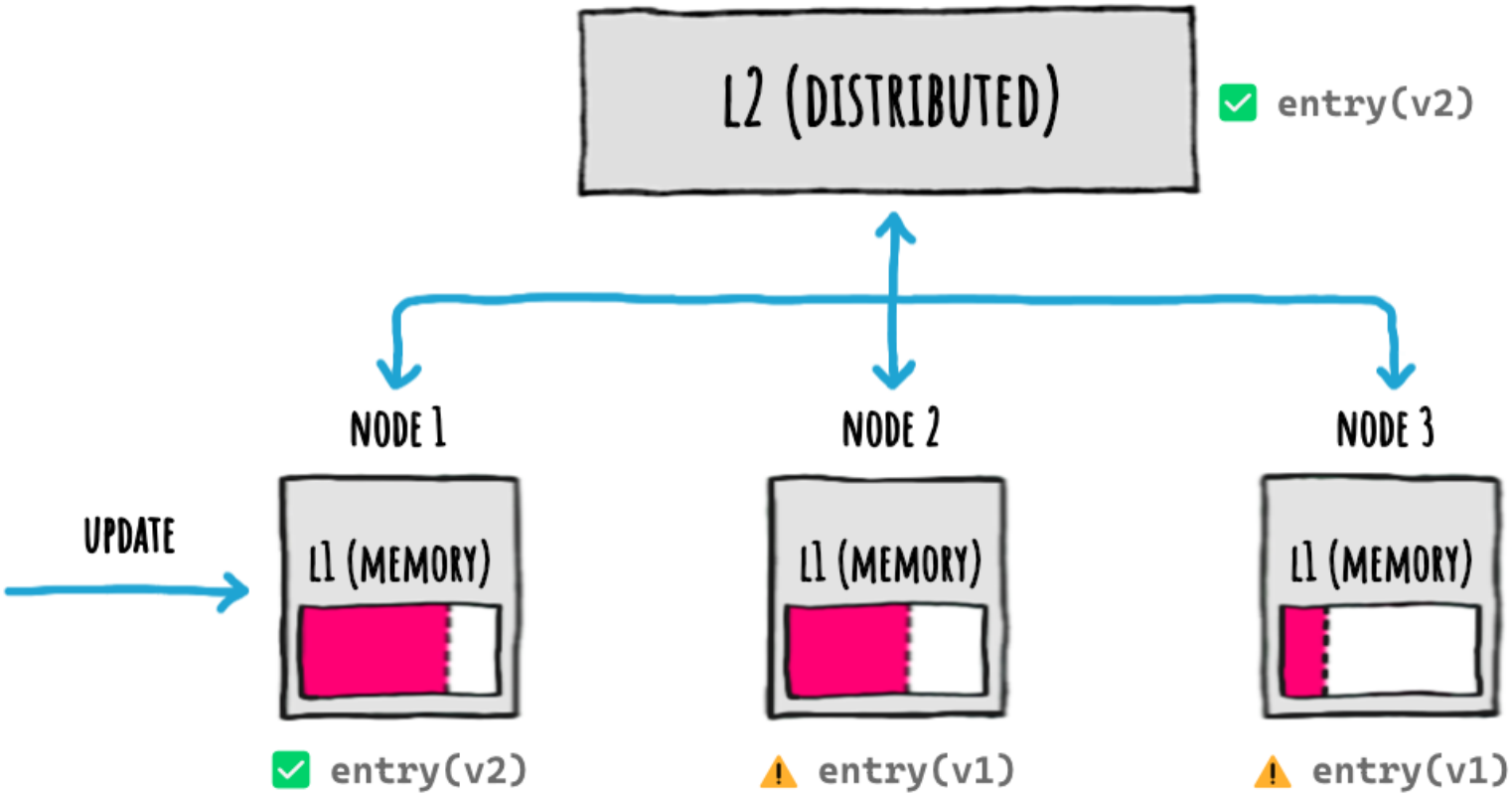
But if other nodes **already** have that **entry cached** in their L1 (memory), what then?

We would need to wait the **expiration** in their own L1.

But...

😨 Cache Coherence

In the time window **before they expire**, this happens:



😨 Cache Coherence

Basically, the cache as **a whole** becomes **incoherent**.

And this is **bad**.

So? What can we do?

Backplane (FusionCache)

📢 Backplane

We can simply use a **Backplane**, to let nodes communicate.

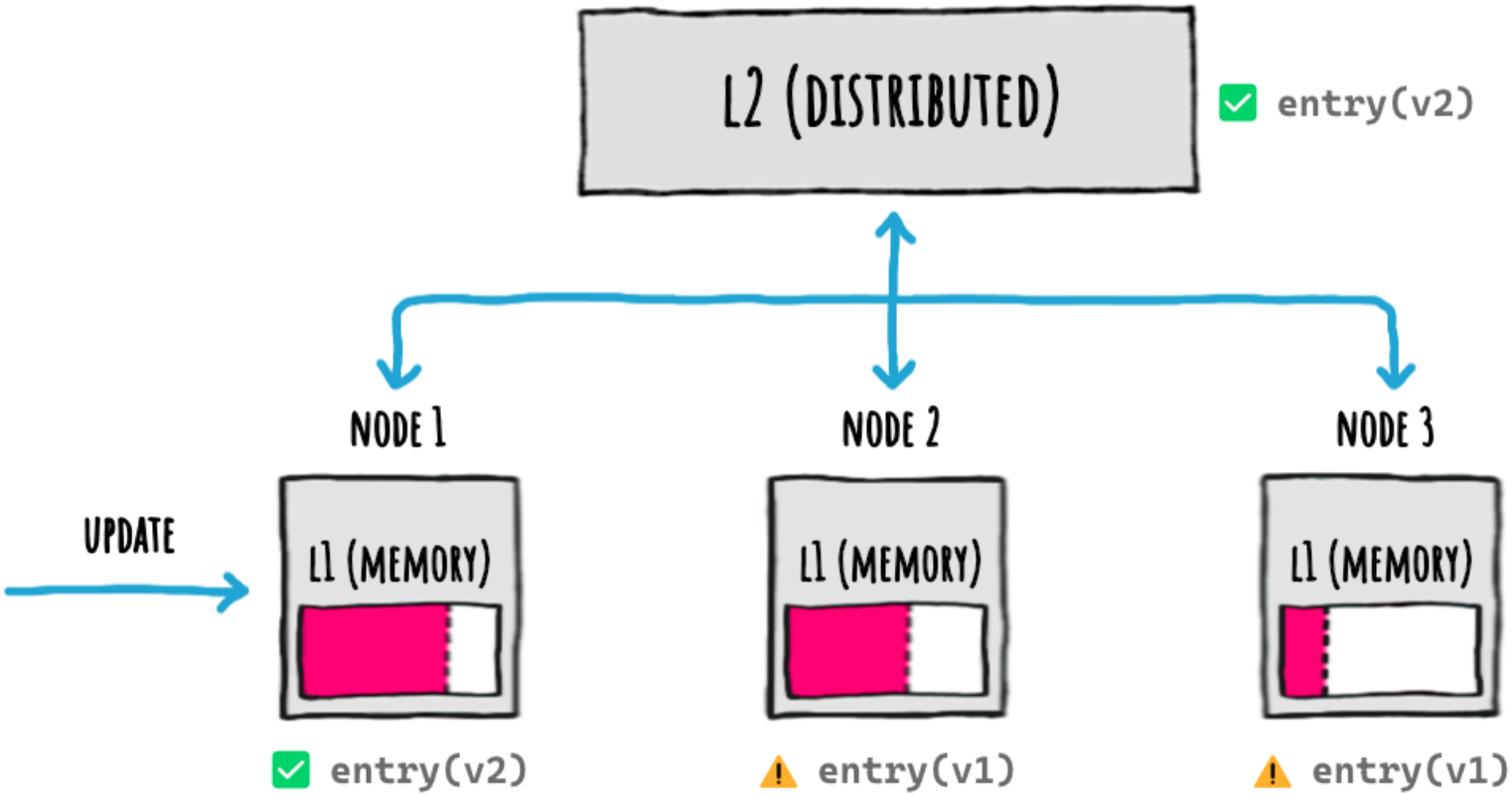
Like this (again, only during setup):

```
services.AddFusionCache()
    .WithBackplane(
        new RedisBackplane(new RedisBackplaneOptions { ... })
    )
;
```

Done 🥳

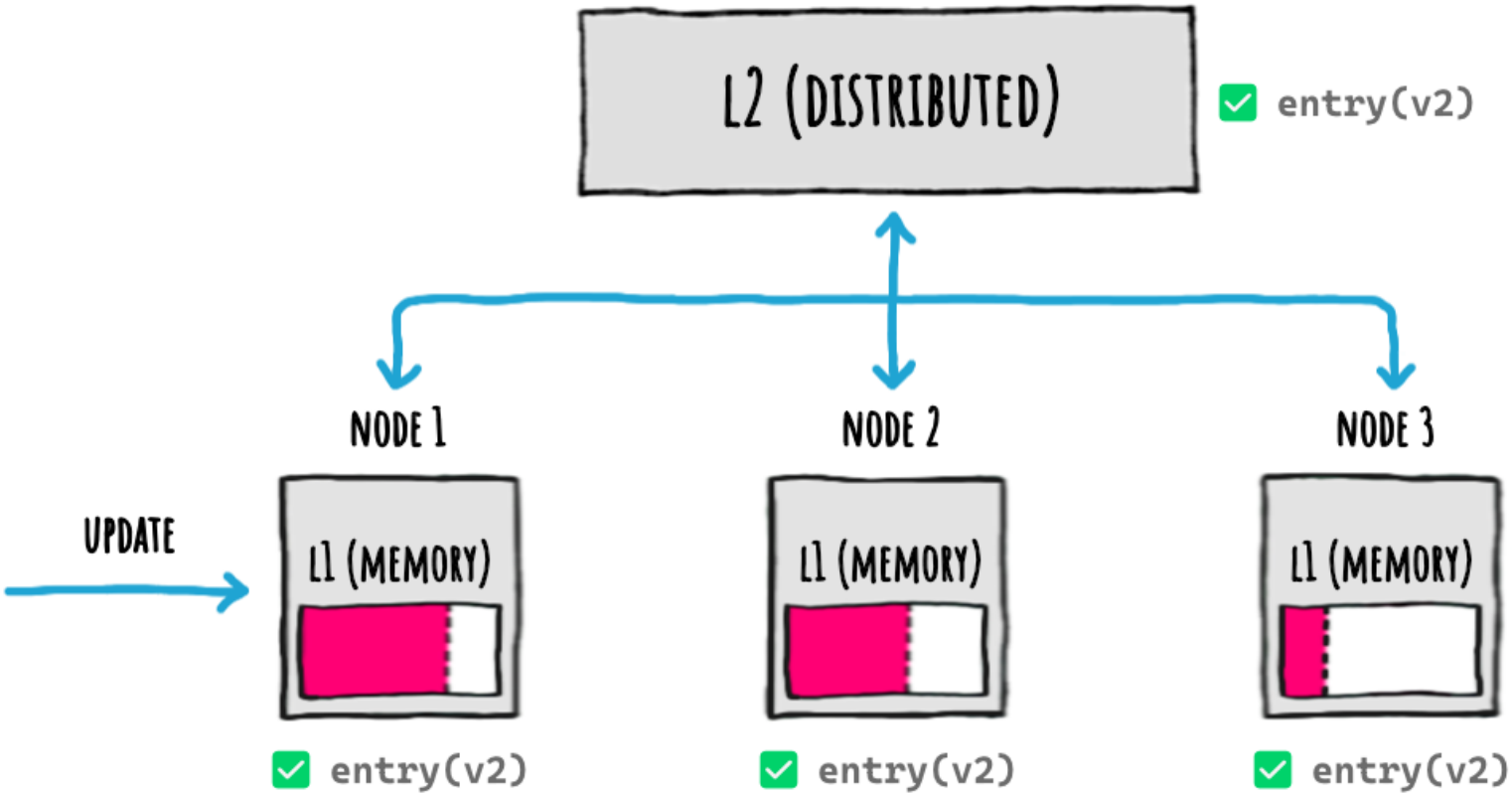
Backplane

And we'll go from this (**incoherent**):



📢 Backplane

To this (**fully coherent**):



Microsoft HybridCache

HybridCache

Early 2024, **Microsoft** announced their own **HybridCache**.

You probably already heard about it, right?

Well, as Íñigo Montoya would say: it may not be what you think it is.

Let’s see...

HybridCache

The design:

- **L1:** uses `IMemoryCache`
- **L2:** uses `IDistributedCache`
- unified serialization interface for `byte[]` (for L2)

It transparently handles:

- one or two **levels**
- one or more **nodes**... kind of (*)

FusionCache design validated 🥳

HybridCache

Other features:

- Cache Stampede protection (*)
- Tagging (*)
- DI + fluent builder (*)
- targets .NET Standard 2.0 (runs everywhere, old + new)

HybridCache

But most importantly, it's both:

- a **shared abstraction**
- a **default implementation**, by Microsoft

In fact:

- **abstraction:** `public abstract class HybridCache`, released with .NET 9
- **implementation:** `internal class DefaultHybridCache`, released with separate package

Think of the `HybridCache` abstract class as the `IDistributedCache` interface: an **abstraction**, but for generic hybrid caches.

HybridCache

How to **register** it (after installing the package):

```
services.AddHybridCache();
```

HybridCache

How to **use** it:

```
public class ProductController : Controller
{
    HybridCache _cache;

    public ProductController(HybridCache cache)
    {
        _cache = cache;
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<Product>> Get(int id)
    {
        return await _cache.GetOrCreateAsync<Product>(
            $"product:{id}",
            async _ => GetProduct(id),
            options
        );
    }
}
```

HybridCache

How to **use** it:

```
public class ProductController : Controller
{
    HybridCache _cache;

    public ProductController(HybridCache cache)
    {
        _cache = cache;
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<Product>> Get(int id)
    {
        return await _cache.GetOrCreateAsync<Product>(
            $"product:{id}",
            async _ => GetProduct(id),
            options
        );
    }
}
```

HybridCache

I shared with the team suggestions, ideas, gotchas, etc and they **listened**: awesome 🎉

Thanks Marc Gravell & team!

All in all, I think the HybridCache effort is a great example of what it may look like when **Microsoft** and the **OSS community** have a constructive **dialogue**.

So, are you using HybridCache already? Maybe in production?

Ok, pay attention...

HybridCache Limitations & Issues (mid 2025)

HybridCache Limitations & Issues

Since the current Microsoft implementation is their very **first version**, it still has some limitations and issues.

Most of them are **not** related to the **abstraction**, but only to their current **implementation**.

Some are minor, some are more severe, so it's important to know them.

Let's look at some of them.

HybridCache: no new()

As we saw the concrete class is **internal** and therefore it's not possible to directly create an instance of it via `new()`.

There's a builder, but that's also **internal**.

This means relying only on the **DI** approach: maybe good, but important to know.

HybridCache: no DI control

About the DI approach, we **cannot** specify what to do with:

- **L1** (memory level)
- **L2** (distributed level)

Both picked up automatically, no control over it.

For example, for L2:

- if an `IDistributedCache` is registered, it **will** be used
- if an `IDistributedCache` is not registered, **nothing** can be used

HybridCache: single instance

Again, about DI: we also **cannot** register multiple named caches.

This means every consuming code in our app will share the same instance: we should be careful about potential **cache key collisions**.

It also means **no different configurations**, since there's only one instance.

HybridCache: async only

The API surface area is **async only**.

This means we can't use HybridCache in non-async call sites, unless we do really bad things like `.Result` or `.GetAwaiter().GetResult()` which... I mean.

HybridCache: no get methods

There are no read-only methods, so it's not possible to just "get a value".

At first it may look like it would be possible to create a custom extension method that simulates it by calling `GetOrCreateAsync()` with some specific entry options:

`github.com/dotnet/extensions/issues/5688#issuecomment-2692247434`

Problems are:

- in a **L1+L2** setup it would **not work** correctly (no L2-to-L1 copy)
- stampede protection is **non-deterministic** (on cache miss)

HybridCache: non-deterministic

This is pretty subtle.

When calling `GetOrCreateAsync(key, factory)`, the expectation is:

- if the value **is** in the cache (cache hit) just **return it**
- if the value **is not** in the cache (cache miss) **execute the factory**

But: currently, a factory may **not run** even when a value **is not** in the cache.
And there's no way to know it.

Basically, the behavior is **non-deterministic** (on cache miss).

HybridCache: non-deterministic

For example, to simulate a get-only method we can try to do this:

- declare a variable `bool found = true`
- call `GetOrCreateAsync<T>()`
- in the factory sets `found` to `false` and returns `default(T)`
- disable writing to L1 & L2

Since a value is always returned, the `found` variable is used to differentiate between:

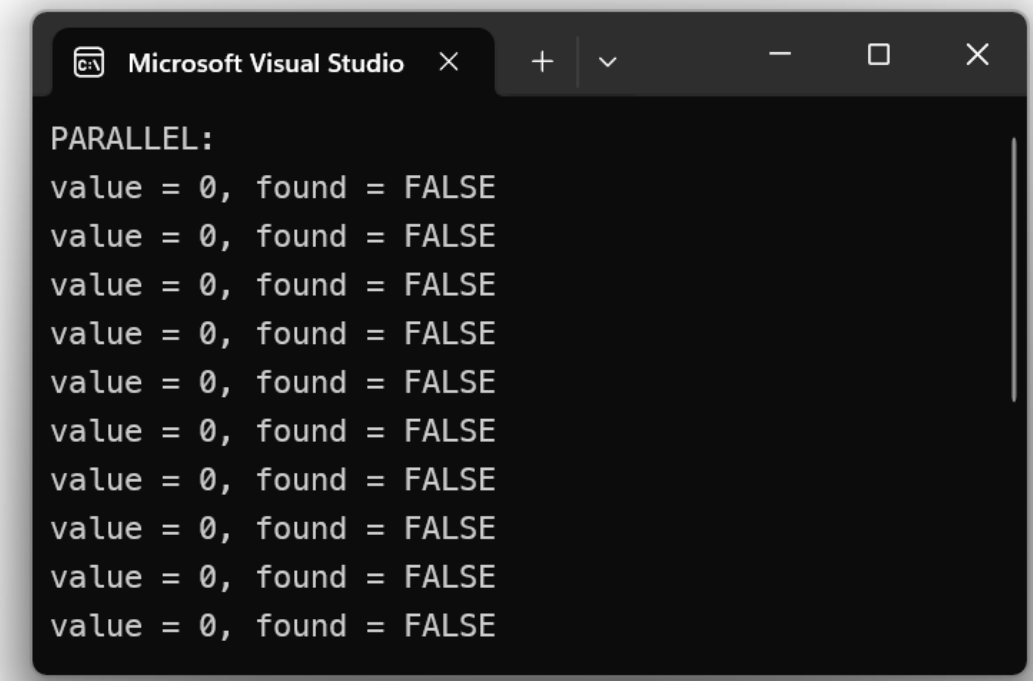
- **not in the cache:** `found == false`
- **in the cache:** `found == true`

Makes sense, right?

Well...

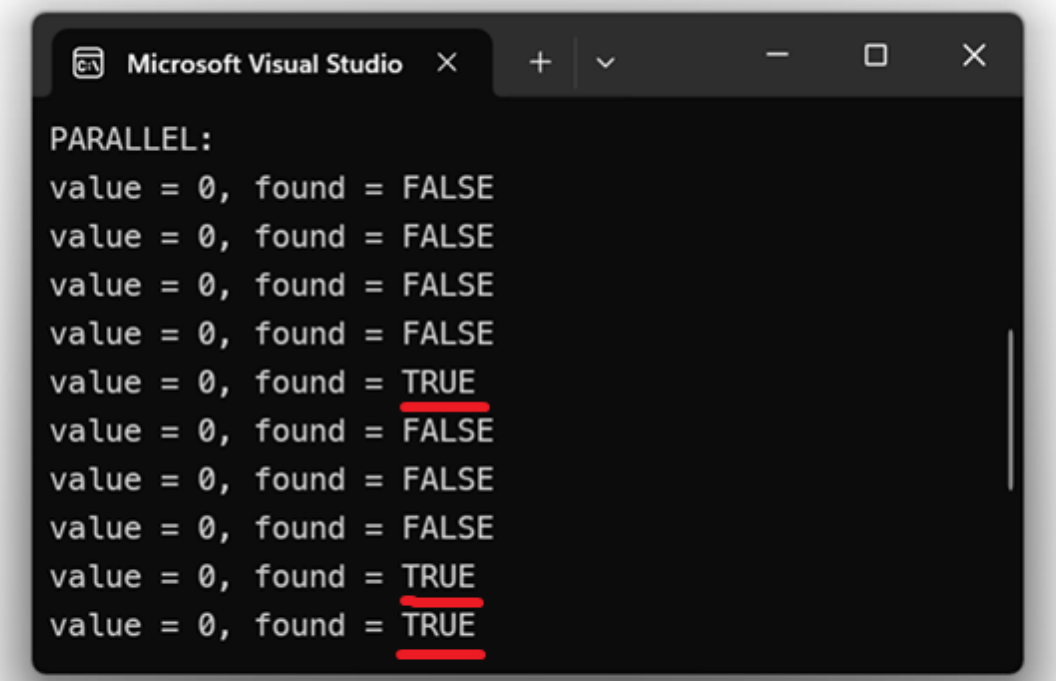
HybridCache: non-deterministic

When executed **in parallel** you would expect this (with `int` as type):



HybridCache: non-deterministic

... but **instead**, you'll get this (it changes every time):



HybridCache: distributed issues

HybridCache currently doesn't have something like the **Backplane** in FusionCache.

So, it **cannot notify** the other nodes about updates, removes, tagging, etc leaving the cache, as a whole, **incoherent**.

In short: because of these limitations, the current Microsoft implementation of HybridCache is, in practice, **not usable** with more than 1 instance/node (imho).

ℹ️ **NOTE:** you may use a lower `LocalCacheExpiration`, as a mitigation.

HybridCache: a solution?

But wait a minute.

We said that:

- HybridCache is also an **abstraction**
- so, other **implementations** are possible
- and **FusionCache** is a hybrid cache

So...

FusionCache as HybridCache

FusionCache as HybridCache

FusionCache is its own, independent thing, and this does not change.

But there may be **value** in being able to work with a **shared** 1st party Microsoft abstraction.

So FusionCache can **also** be used as an implementation of the new `HybridCache` abstraction, while keeping the **extra features** of FusionCache.

And all thanks to a small **adapter**.

How?

FusionCache as HybridCache

Like this:

```
services.AddFusionCache()
  .AsHybridCache(); // MAGIC
```

No code changes required, anywhere.

FusionCache as HybridCache

Remember those (*) ?

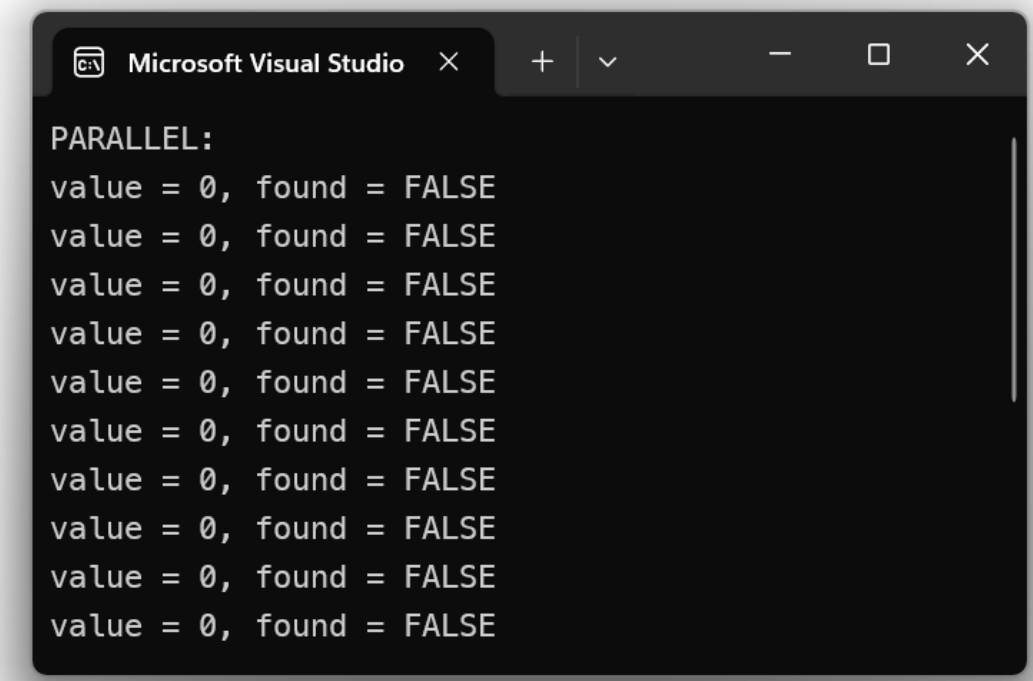
Gone 👋

Using FusionCache as an implementation of HybridCache allows:

- **stampede protection:** deterministic + unified + sync/async
- **control:** total control over L1/L2 setup
- **backplane:** instant updates on multiple nodes
- **no distributed issues:** bye bye out-of-sync nodes and cache incoherence
- **more features:** Fail-Safe, Eager Refresh, Factory Timeouts, Auto-Recovery, etc
- **multiple caches:** we can use Named Caches with Keyed Services

FusionCache as HybridCache

About the non-deterministic problem, now it's like this:

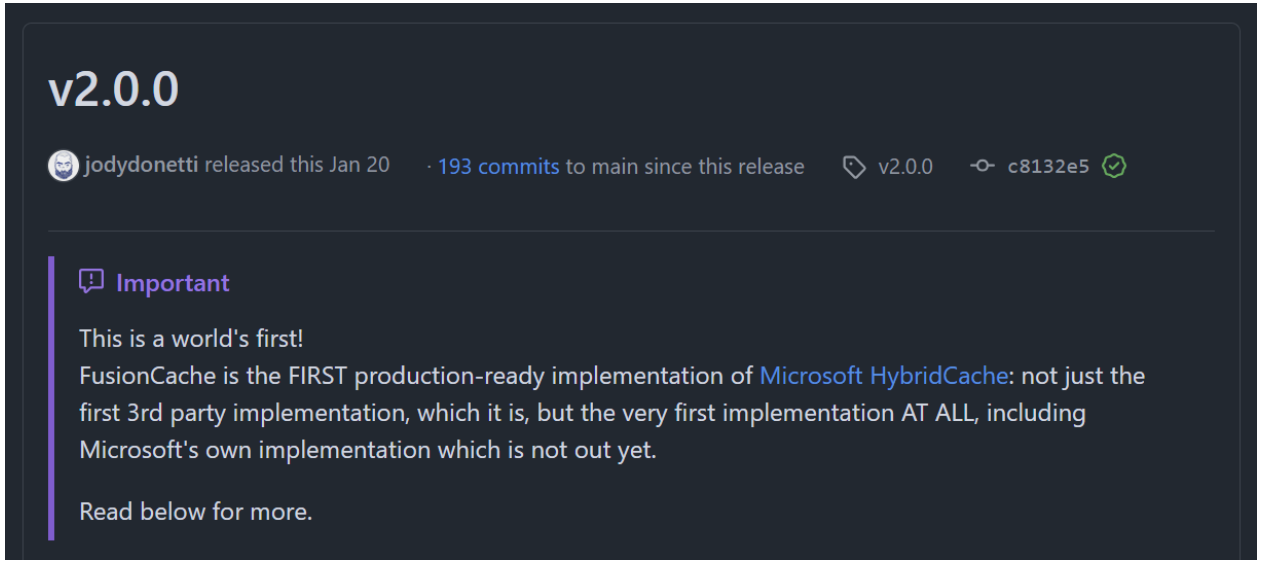


FusionCache as HybridCache

The timeline:

- **Nov 2024:** HybridCache **abstraction** released (with .NET 9)
- **Jan 2025:** FusionCache v2 released (with **HybridCache adapter**)
- **Mar 2025:** HybridCache **default implementation** by Microsoft released

World's first



Recap

⚡ Memory Caches

Pros/Cons:

- ✅ **easy:** easy to use
- ✅ **data locality:** near our own code, in the same memory space
- ✅ **cost:** no remote calls, no (de)serialization
- ✅ **availability:** it is always available
- ✅ **stampede protection:** maybe there (check the library)
- ⛔ **cold starts:** at every restart the cache is empty
- ⛔ **horizontal scaling:** data is not shared between multiple nodes

✈️ Distributed Caches

Pros/Cons:

- ⛔ **easy:** less easy, we need to handle (de)serialization, etc
- ⛔ **data locality:** data is out-of-process, typically remote
- ⛔ **cost:** network calls + (de)serialization
- ⛔ **availability:** may not be always available (or reachable)
- ⛔ **stampede protection:** not there
- ✅ **cold starts:** at every restart the cache is already populated
- ✅ **horizontal scaling:** data is shared between multiple nodes

🚀 Hybrid Caches

Pros/Cons:

- ✅ **easy:** easy to use
- ✅ **data locality:** near our own code, in the same memory space (**L1**)
- ✅ **cost:** no remote calls, no (de)serialization (**L1**) except for the first L1 cache miss
- ✅ **availability:** it is always available (**L1**)
- ✅ **stampede protection:** usually there (but always check the library)
- ✅ **cold starts:** at every restart L1 is empty, but **L2** is already populated
- ✅ **horizontal scaling:** data is shared between multiple nodes (**L2**)

Also, a single **unified** API, and with more **features**.

The best of both worlds.

So, hybrid caching: cool?

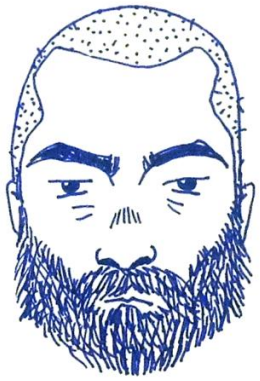
😎 So, hybrid caching: cool?

To go **hybrid**, we can:

- use **FusionCache**, directly
- use **HybridCache** abstraction, with the **default** implementation by Microsoft (*)
- use **HybridCache** abstraction, with the **FusionCache** implementation

We have choices, let's make the most out of them 🙂

Hvala! Thanks!



github.com/jodydonetti

twitter.com/jodydonetti

linkedin.com/in/jody-donetti

WE APPRECIATE YOUR FEEDBACK,
AND IT MAKES US BETTER EACH YEAR.

YOU CAN COLLECT ADDITIONAL NT COINS
BY FILLING OUT THE QUESTOINNAIRES.

THEN VISIT GEEK BOUTIQUE AND
SPEND NT COINS ON COOL GIVEAWAYS.

NT 3.0 KONF


NT
30
KONF