2023 OFA Virtual Workshop

Supporting an Upstream First Kernel Driver for HPC Fabrics

Dennis Dalessandro, Manager Software Engineering (Linux Kernel)

Cornelis Networks

Notices and Disclaimers

CORNELIS™ NETWORKS

Obligatory Intro Slide

- What is Cornelis Networks?
    - Omni-Path Architecture (OPA)
    - Third year of talks in this workshop
    - Spun out of Intel
- What is OPA?
    - Low latency, high speed fabric interconnect
    - Current 100Gbps
    - Next gen 400Gbps, and beyond
- Who am I?
    - Lead Kernel Development team at Cornelis Networks
    - One of original developers of our (hfi1) driver technology

This talk is...

- Not about licenses
  - I'm not a lawyer
  - Everyone has personal and company-oriented views
- Not a discussion on hfi1 architecture
  - Will talk about history of how it made it into the kernel
- Not strictly applicable to Cornelis Networks or hfi1
  - Concepts apply to other drivers
  - Concepts apply to other software as well
  - Not a guide on upstreaming drivers rather a call to make it a priority

Why Open Source?

- Think we can skip the “Why Linux” discussion?
- Advancement of technology
- Benefit of thousands of people looking at your code
- Competitors actually work together for the common good
- This is the only way to make it into distros
- Customers want to know what code is being run
- It’s just easier on customers if they don’t need to install stuff
- Path to ubiquitousness

Drawbacks to Open Source

- Kernel code is not yours anymore
    - Community “owns” it
- Can’t toss whatever code you want in
    - Ensures higher quality
    - But some “features” may not be accepted
        - I’ve got examples!
- Timeline is drawn out
    - Kernel.org for instance follows a pretty steady cadence

Why Kernel.org?

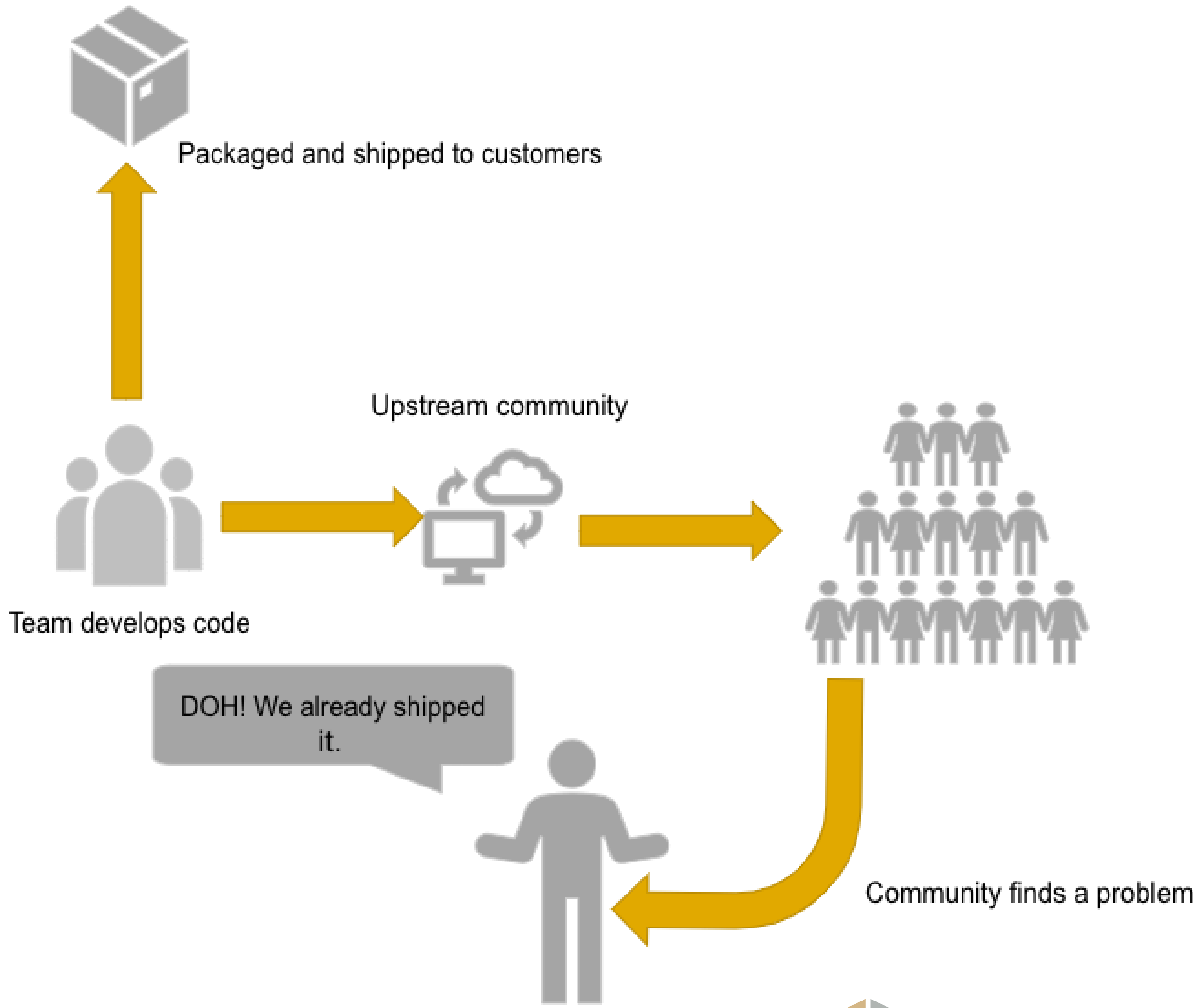
- Customers want this
- Distros require it
- The community is HUGE
- When dealing with this community, companies…
  - Need a way to get hot fixes to customers
  - Need a way to deploy features not appropriate for kernel.org
  - Need a way to issue debug/test code
  - Need a way to follow their own release schedule
- How do we meet these requirements?

Follow Upstream First Principle

- Upstream = Linux Kernel.org for my examples
    - Could apply to libfabric, any other **community driven** project
- Note does not necessarily mean Upstream Only
    - Would be an ideal model but not realistic for most things
    - Mature products that aren't changing or adding new features are easier

If we don’t upstream first?...

- Worst case
  - Customer has a feature that we have to take away
- Best case
  - Diverging code lines to deal with
- Either way quality suffers
- Customer experience not ideal
- Not fool proof
  - Code can be accepted, and bugs found long after acceptance
  - This is one of the advantages of OSS
  - Goal is to catch problems early!


Packaged and shipped to customers
Upstream community
Team develops code
DOH! We already shipped it.
Community finds a problem

CORNELIS™ NETWORKS

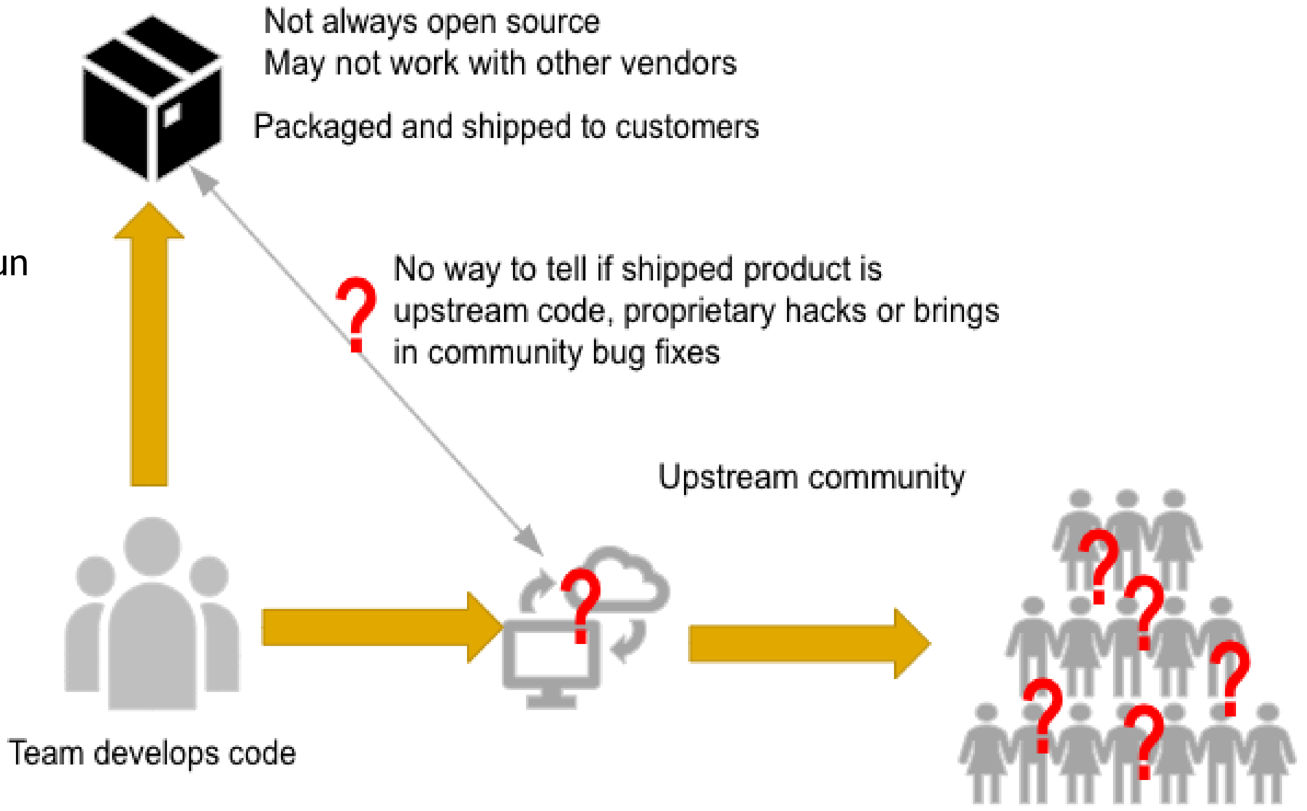
Where are we now?

- Described what Open Source is and its pros/cons
- Talked about Upstream First and consequence of not doing it

- Next: Approaches to software release

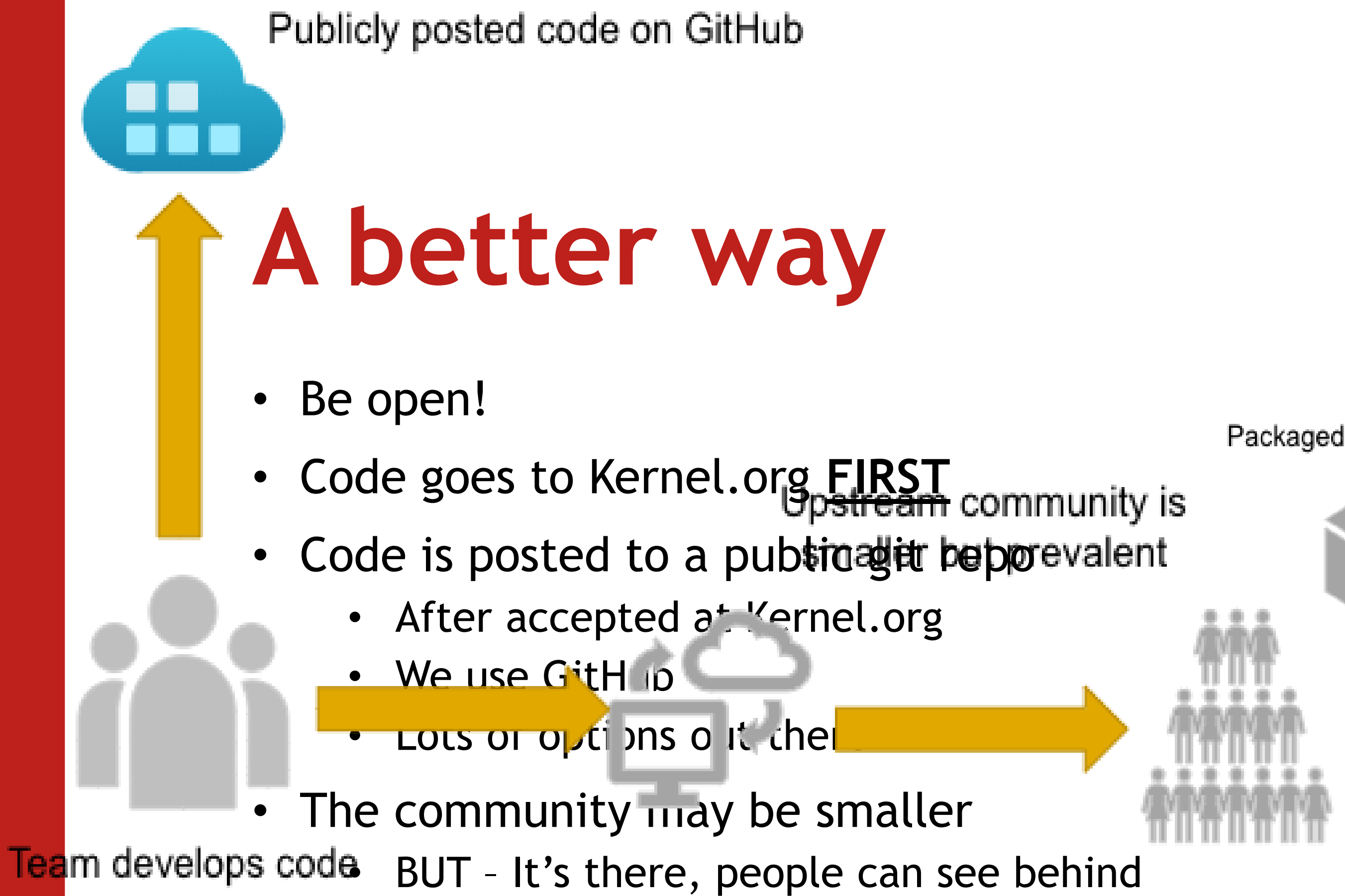
Black box approach

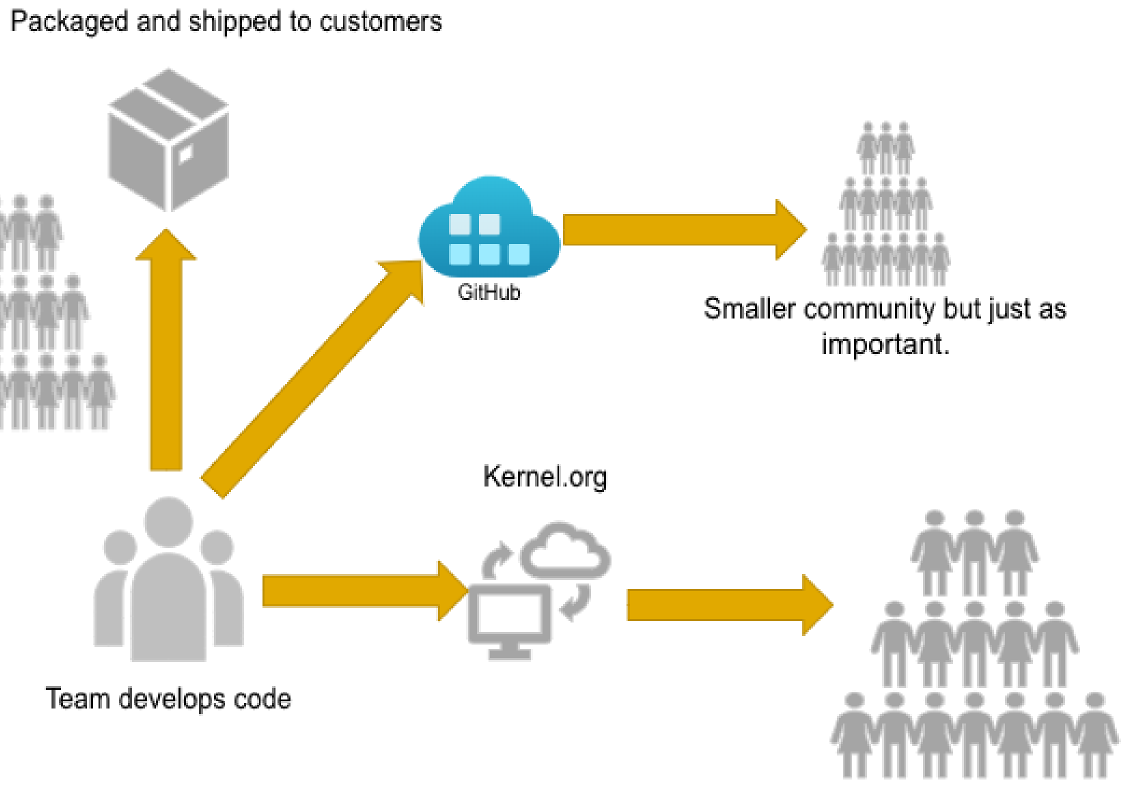
- Instead of working with code upstream
  - Just make it a block box
  - Maybe ship some stuff upstream
    - No one knows if it is what customers run
  - Get to release whatever you want
- Unfriendly to users
- Unfriendly to the community
- Does not better technology


Not always open source
May not work with other vendors
Packaged and shipped to customers
No way to tell if shipped product is upstream code, proprietary hacks or brings in community bug fixes
Upstream community
Team develops code

A better way

- Be open!
- Code goes to Kernel.org **FIRST**
- Code is posted to a public git repo
  - After accepted at Kernel.org
  - We use GitHub
  - Lots of options out there
- The community may be smaller
  - BUT – It's there, people can see behind the curtain
  - It is its own "upstream"
- Stuff going to customers
  - Accepted upstream first!


Publicly posted code on GitHub
~~Upstream~~ community is smaller but prevalent
Team develops code


Packaged and shipped to customers
GitHub
Smaller community but just as important.
Kernel.org
Team develops code

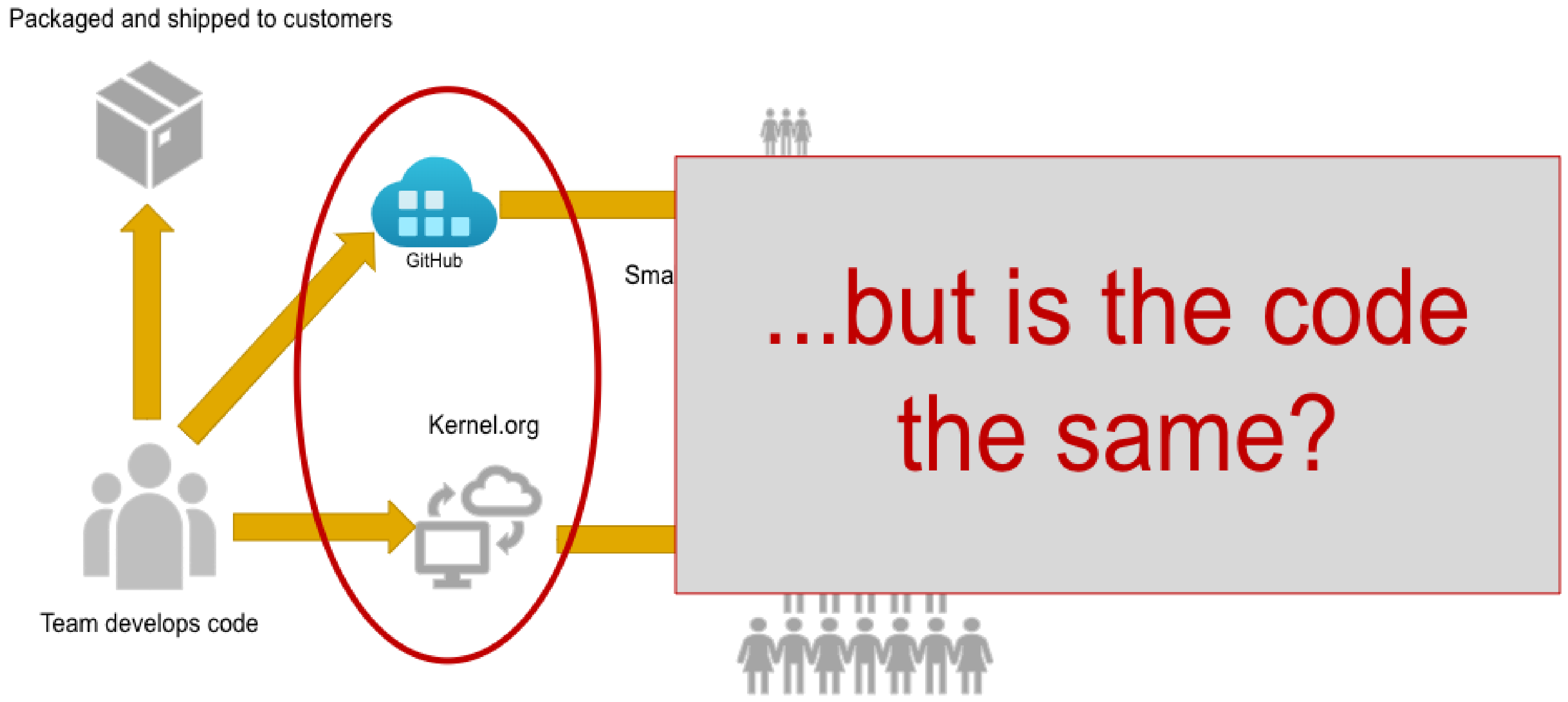
CORNELIS™ NETWORKS

Where are we now?

- Described what Open Source is and its pros/cons
- Talked about Upstream First and consequence of not doing it
- Covered two approaches to software release
  - Should be clear which one is better

- Next: Dealing with code bases
  - Kernel.org
  - GitHub
  - Distro

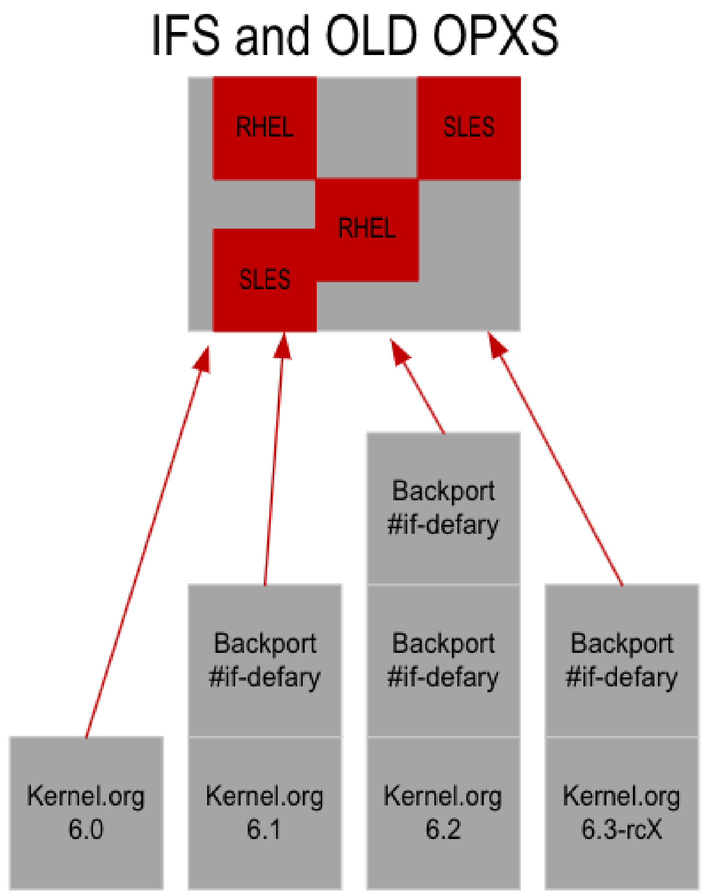
Open is great and the right way...



...but is the code the same?

Backport the world to one code base

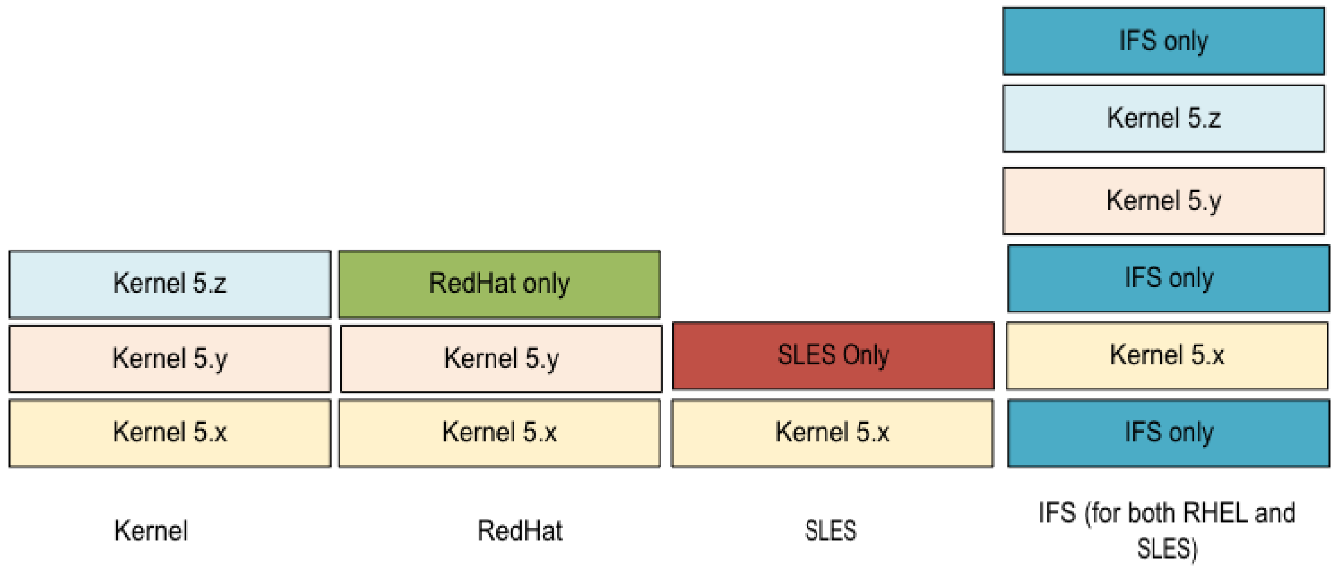
- For us: One code base with compat code for different distros
- Lots of #if-defs
- Maintain a single release stream
  - As time marches on back port changes made to driver
  - Backport goal is to work with distro kernels
    - Usually current and one-back
  - Backport work is HARD
    - Especially when new APIs are introduced
- Still accomplishes the goal of being open and transparent
- This is what Intel IFS and early Cornelis OPXS did
  - It works! It's tried and true.
  - Accomplishes our open source goals
  - Gets the customer going with the distro they choose
  - *Maybe there is an even better way...*



Another way to look at the current state

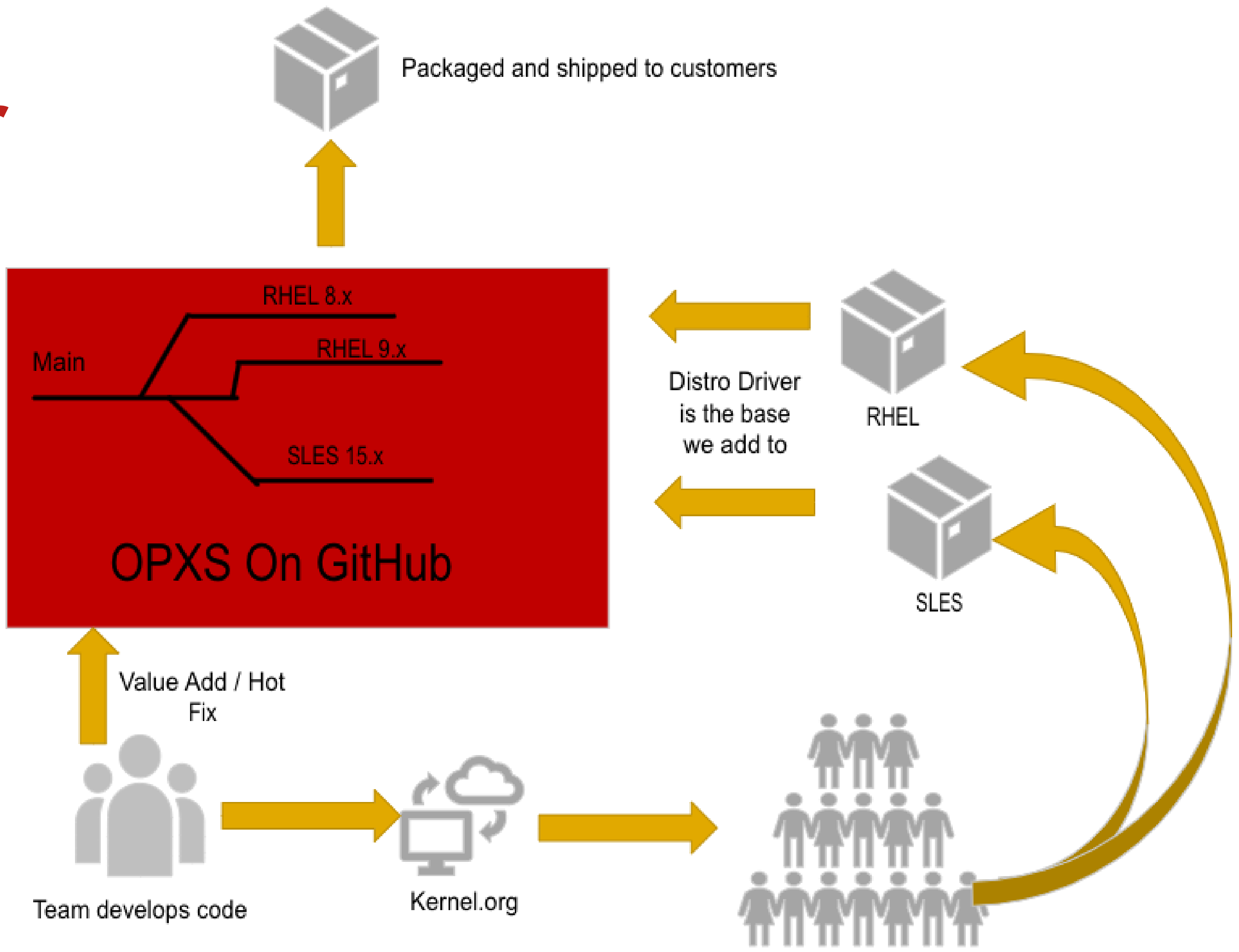
- Applies to IFS or old OPXS
- Everything is totally different
- In-box distro driver VASTLY different than backport driver
- We must FULLY validate:
  - Kernel.org
  - Each Distro in-box
  - Each Distro with IFS/OPXS
- A lot of validation
- More code = More risk
- Customers want it to “just work”
- We are duplicating work that distros are experts at

IFS code is just different. Over time that difference grows larger. IFS is continually diverging. It does not ever "reset" or "sync" up with the distros. Continually adopts Kernel.org.

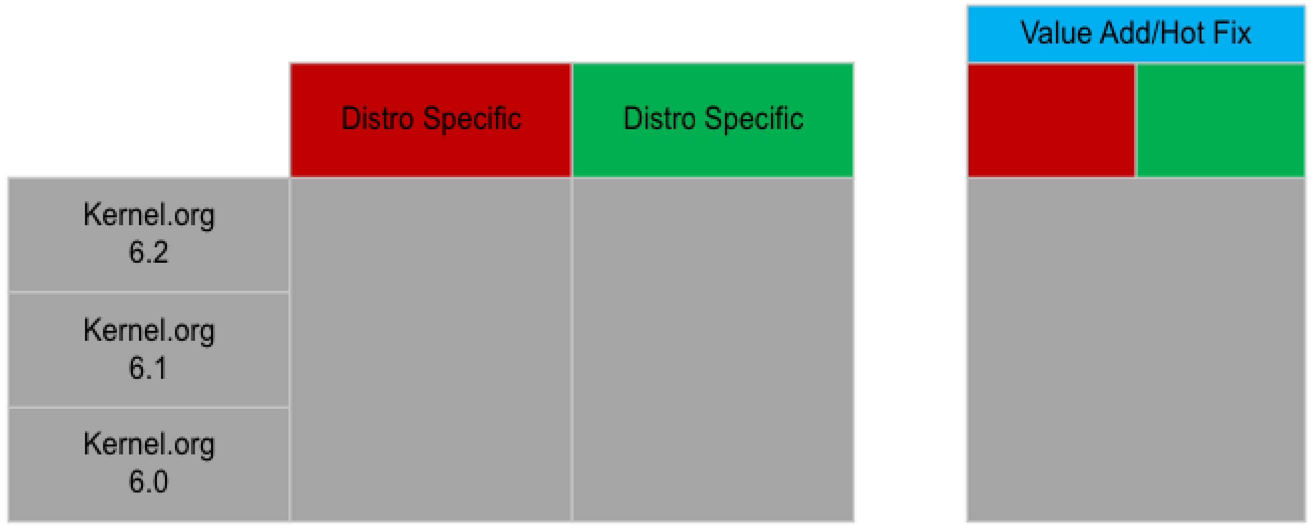


How we do better

- Let the distros do what they do
- Leverage distro code and add to it
    - Diagnostics not suitable for upstream
    - Hot Fixes between distro releases
- Distro pulls code in from Kernel.org
- So, FEED Kernel.org our code!



Ideally, the end result



OPXS = Distro Driver + A little bit

What does it take?

- **Cornelis fully embraces upstream first development**
  - Not just lip service
- Validation effort and CI on upstream Kernel.org
  - Upstreaming is not a side task it is THE task
- Validation early and often with Linux distros
  - Helps to partner with distros to get early access for testing
  - Ensures our drivers are fully vetted on each distro, **BY THE TIME IT GA's**

What about “new” stuff?

- New technology is harder
- Must balance upstreaming code with keeping things secret
  - Upstream what you can, when you can
  - **All parts of company must buy in to upstream first not just the ones writing the code**
- With a good foundation in Kernel development and upstream process its possible
  - Learn from previous stumbling bocks
  - Companies need to prioritize experience in upstreaming code
    - It is not go find engineers that can make the stuff
    - It is go find engineers that can make the stuff **and** get it upstream
- Days of upstream development being a secondary or ancillary task are OVER
- Those who collaborate and work in the community prosper
- Those who choose to isolate will wither

Doesn't the kernel move slow?

- Not really
- How many companies push out a software release every 8-10 weeks like clockwork?
  - Not very many, but Linux does
- But the distros increase the lag
  - Fair point, they do, but we have a path to provide hot fixes/updates
    - If code is accepted it's generally safe to flow into our release and code base
      - Why? Because UPSTREAM FIRST!
      - Recall: not upstream only
    - And only those fixes/updates, does not need to be a full-blown new release
  - Remember customers WANT the stableness of a distro
  - Distros have z-stream and security updates regularly
    - Severe issues we can appeal to distros
- Keep the flow of code going upstream, distros will pick it up

Where are we now?

- Described what Open Source is and its pros/cons
- Talked about Upstream First and consequence of not doing it
- Covered two approaches to software release
  - Should be clear which one is better
- Went over dealing with code bases (Kernel.org, distro, OPXS)
- Looked at how we do upstream first

- Next: Story Time

hfi1: A story to not be repeated

- hfi1 driver has been through a lot of ”change”
- started out as a literal copy of the qib driver (QLogic IB driver)
  - similar HW, its a good framework, so why not? This isn’t the issue
  - **<u>Problem</u>**: verbs code between qib and hfi1 is basically identical
    - There is A LOT of verbs code, a lot
    - Oh yeah also another copy in the ipath driver! So 3 copies!
- Upstream community had a pretty visceral reaction
  - Partly due to competitive companies
    - Things are VERY different today, kernel.org is a unique community
  - However, there was a lot wrong with the driver
    - TWSI, Multiple cdevs, snoop/capture, Register debug, etc....
    - Point is we had a list, yes a literal list
    - Made deal to excise that stuff and go to staging
      - This was a HUGE headache and plagued us for years to come

Why?

- We had brilliant engineers
  - Amazing coders
  - Knew the hardware in and out
  - Expert knowledge of how to program in a kernel environment
- We did not have upstream expertise
  - No one had upstreamed a driver before
  - qib was grandfathered
- Lesson Learned (Companies pay attention):
  - You NEED people with exp in their community
    - Must do the work and pay your dues
  - You NEED to realize the importance of the upstream community
    - Too often it is looked at as a side project
  - **<u>Botom line: Needs to be a priority</u>**
    - All too often it is not

Circling back to Upstream First

- Want to keep the stuff we cut... Hurray for IFS!!
- Now we have:
    - distro drivers (but we are in staging so limited to tech-preview)
    - upstream drivers
    - IFS – tested, tried, and true, reliable workhorse
        - This is the “supported” way
        - Bells and whistles upstream doesn’t have
- Over time:
    - Code diverges,
    - Upstream fixes bugs
    - New APIs need reconciled
    - In the exact position we don’t want to be in (see the previous slides!)

How did hfi1 overcome?

- **<u>Simple: We started doing upstream first</u>**
- A long time ago, in a conference room far, far away…
  - (Ok like 8 years ago in King of Prussia, PA)
  - Met with upper mgmt and convinced them this is the right course
  - Explained how upstream first:
    - Saves time
    - Increases code quality
    - Increases customer satisfaction
    - Encourages growth of the technology
  - There was some kicking and screaming
  - In the end, we are in the right place going the right direction

Key Takeaways:

- Open Source is key
- Upstream First is the way forward
    - Upstream first can work to
        - Improve code quality
        - Decrease time to market
        - Keep customers happy
- Nothing is absolute
    - May be times when upstream first just doesn't work (CVE?)
        - This should NOT be the goto and avoided at all costs
    - It's a goal
- Cornelis Networks is fully embracing Upstream First as a key part of our approach to openness
- Cornelis hfi1 driver development is a real-world example of upstream first development
    - Other efforts following suit, like OPX and Libfabric
        - More on that in the next talk

Thank You

www.cornelisnetworks.com