Heterogeneous Communication API For eHPC (embedded High Performance Computing)

https://github.com/michaelboth/Takyon

Michael Both: May 23, 2023 (updated Dec 16, 2025)

What is Takyon?

It’s a modern message passing communication API focused on eHPC (e.g. edge computing)

Key Features:

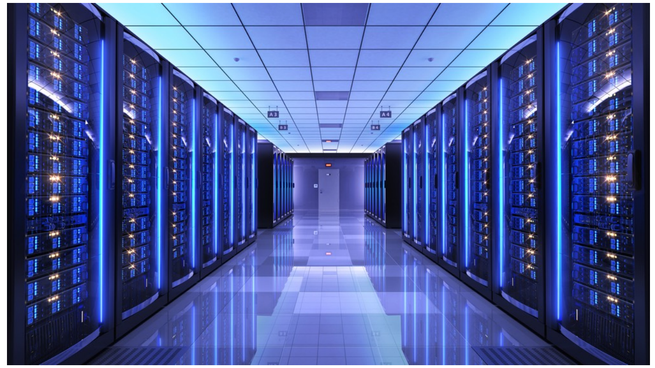
- Reliable and unreliable (including multicast) communication
- Most interconnects: e.g. RDMA, sockets
- Any locality: inter-processor, inter-process, inter-thread
- Two-sided (send/recv), and one-sided (read/write)
- Blocking and non-blocking transfers
- Fault tolerant capable (via timeouts, disconnect detection, dynamic path creation)
- Most memory types: e.g. CPU, GPU

Takyon's Intended Audience

Small heterogeneous systems (eHPC, edge computing, SWaP limited environments, etc)



Large homogeneous systems, with highly collective needs are better suited by MPI or libFabric



Why Introduce Another Communication API?

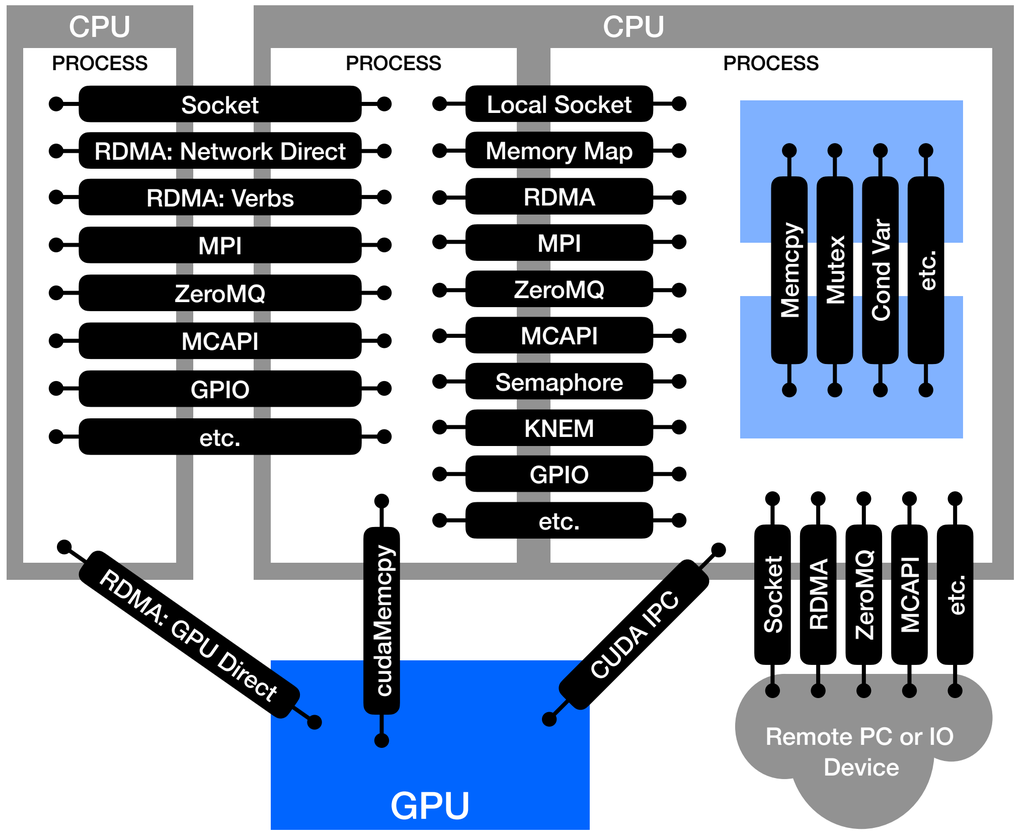
Takyon is an evolution that solves four major communication issues at once:

- One API for all device/interconnect variations
- Supports all common communication features
- Simple and intuitive: 8 functions and 8 data structures
- Best possible performance: throughput, latency, and determinism

The next four slides provide details on the above

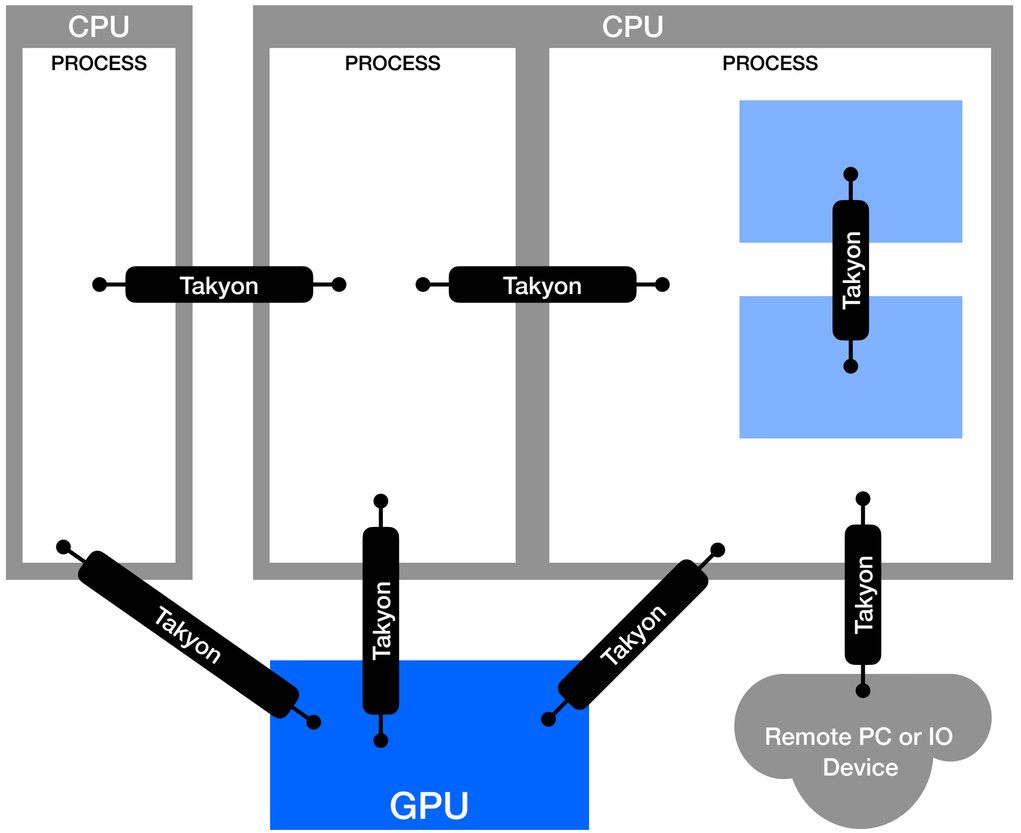
One API for all Device/Interconnect Variations

Before Takyon



No single solution to fit all devices and localities

With Takyon



Supports all Common Communication Features

Feature	Sockets and similar: MCAPI, ZeroMQ	RDMA: Verbs and Network Direct	OFA's libFabric	MPI	Takyon
Reliable and Unreliable	✅	✅	✅		✅
Communication to external apps, sensors, and other IO devices	✅	✅	?		✅
Fault tolerant hooks (timeouts, disconnect detection, path creation)	✅	✅	?		✅
Deterministic: avoids implicit communication and allocations	✅	✅	✅		✅
Includes Inter-thread communication			✅		✅
Non blocking transfers		✅	✅	✅	✅
One way read/write/atomcis: no involvement from remote endpoint		✅	✅	✅	✅
GPU support		✅	✅	✅	✅
Multiple memory blocks per message		✅	?	✅	✅
Memory pre-registered before transfer		✅	✅	Partial	✅
Zero copy and one-way (i.e. no implicit round trip)		✅	?		✅
32bit piggy back message with main message		✅	?		✅

Simple and Intuitive

API	Function Count	Typical Drawbacks
Sockets	~20	Lots of options, confusing terms
RDMA (OFA Verbs, Network Direct)	~100	Overwhelming learning curve, experts are rare
OFA's libFabrics	~100	Overwhelming learning curve, experts are rare
MPI	~300	Large learning curve for various transfer models, limited to reliable communication, 'mpirun' is not portable and can be difficult to tune
Takyon	8	Simple API, intuitive terms and concepts

Takyon eHPC audience is likely focused on something other than communication, such as radar processing, and won't have the time for a high learning curve

Best Possible Performance

Latency, Throughput, and Determinism

- Zero-copy, and one-way (no round trips!)
  - Achieved by:
    - Pre-registering transport memory when the path is created, but before communicating
    - Pre-posting memory for receiving, before communicating
      - Creates a holding place for data to arrive at a later time
      - This makes sure there is no delay or implicit buffering needed when sending
- Non-Blocking
  - Offload transfers to a DMA to allow for efficient concurrent processing and IO

Not all interconnects support the above, but Takyon's abstraction does not inhibit or degrade the interconnects that do support the above.

Takyon API: 8 Functions

	Function	Description
	takyonCreate()	Create one endpoint of a communication path
	takyonDestroy()	Destroy the endpoint
Two-sided functions	takyonSend()	Start sending a message If the communication does not support non-blocking then this will block
	takyonIsSent()	Check if send is complete, up to a specified timeout period.
	takyonPostRecvs()	If supported, pre-post a list of recv requests before the sender starts sending Provides memory buckets for receiving messages asynchronously
	takyonIsRecved()	Check if a message has arrived, up to a specified timeout period.
One-sided functions	takyonOneSided()	Start a one sided message transfer (read, write, atomics)
	takyonIsOneSidedDone()	Check if one-sided transfer is complete, up to a specified timeout period.

Not all interconnects support the above; e.g. sockets don't support one-sided or posting receives

Takyon Provider: Defines the Interconnect

All providers are defined in a text string passed to takyonCreate()

Locality	Examples
Inter-Thread	"**InterThread** -pathID=<non_negative_integer>"
Inter-Process	"**InterProcess** -pathID=<non_negative_integer>” “**SocketTcp** -local -pathID=<non_negative_integer>"
Inter-Processor	"**SocketTcp** -client -remoteIP=<ip_addr> -port=<number>” "**SocketTcp** -server -localIP=<ip_addr>|Any -port=<number> [-reuse]” "**SocketUdpSend** -multicast -localIP=<ip_addr> -groupIP=<multicast_ip> -port=<number> [-noLoopback] [-TTL=<time_to_live>]" "**SocketUdpRecv** -multicast -localIP=<ip_addr> -groupIP=<multicast_ip> -port=<number> [-reuse] [-rcvbuf=<bytes>]” “**RdmaRC** -client -remoteIP=<ip_addr> -port=<number> -rdmaDevice=<name> -rdmaPort=<number>” “**RdmaRC** -server -localIP=<ip_addr>|Any -port=<number> [-reuse] -rdmaDevice=<name> -rdmaPort=<number>” “**RdmaUC** -client -remoteIP=<ip_addr> -port=<number> -rdmaDevice=<name> -rdmaPort=<number>” “**RdmaUC** -server -localIP=<ip_addr>|Any -port=<number> [-reuse] -rdmaDevice=<name> -rdmaPort=<number>” “**RdmaUDMulticastSend** -localIP=<ip_addr> -groupIP=<multicast_ip>” “**RdmaUDMulticastRecv** -localIP=<ip_addr> -groupIP=<multicast_ip>”

No limit to the Takyon Provider possibilities: GPIO, sensors, FPGAs, etc.

Transport Memory

Takyon does NOT allocate transport memory (this is intentional)

Transport memory may need to be shared between communication paths (Takyon or 3rd party) and other processing APIs and IO devices

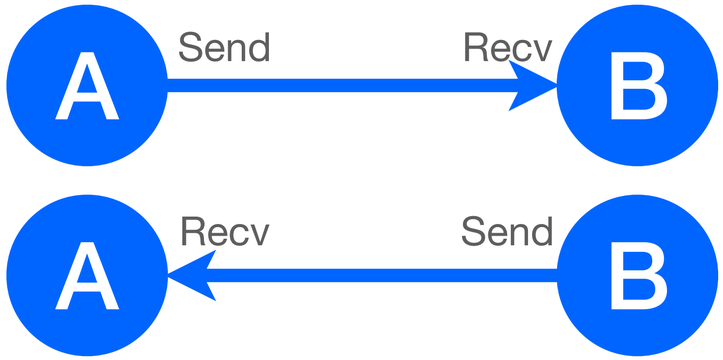
Therefore it is logical to have the application organize all transport memory

- CPU (local or memory map)
- GPU (CUDA)
- Sensor/FPGA/etc. memory

And then provide pointers to the TakyonBuffer structure and other 3rd party APIs

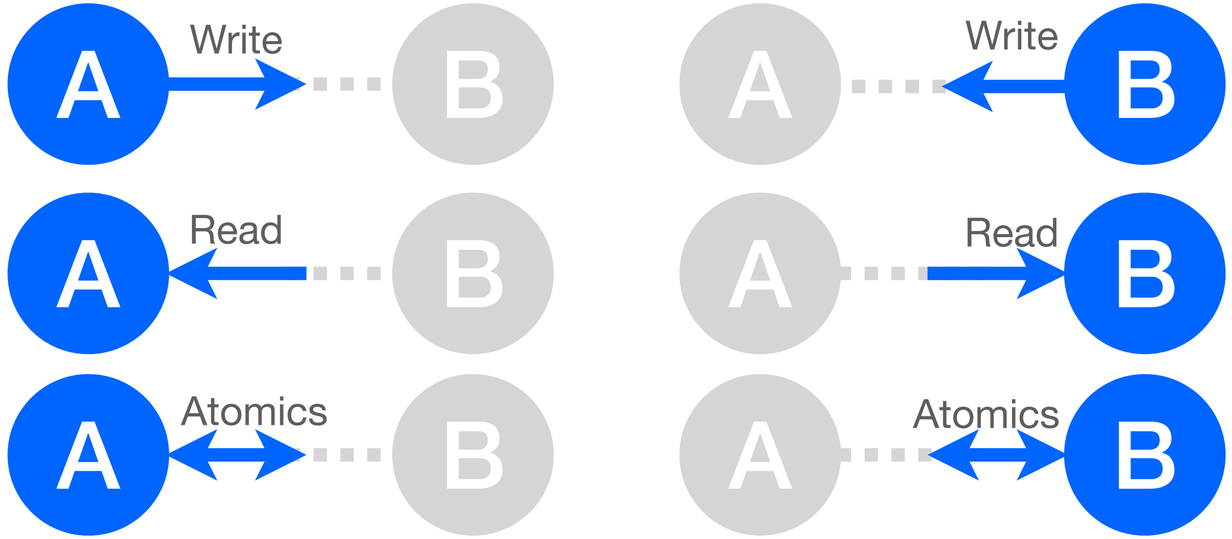
Two-Sided versus One-Sided Transfers

Two-Sided



- Both endpoints are involved with a coordinated send() and recv()
- Unicast and Multicast are two-sided

One-Sided



- Only one endpoint is involved with the transfer
- Can support read, write, and atomics

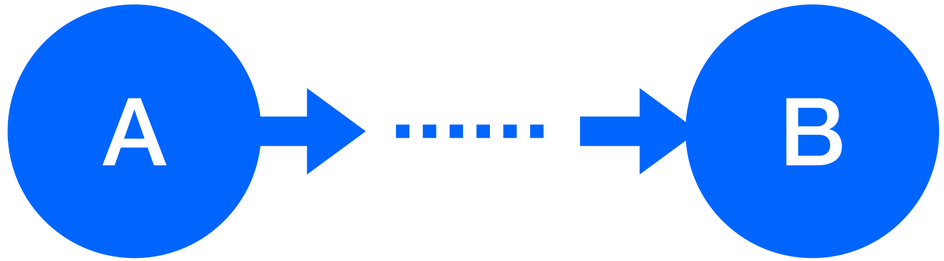
Some Takyon Providers only allow one or the other

Takyon Supports Reliable and Unreliable Transfers

Reliable

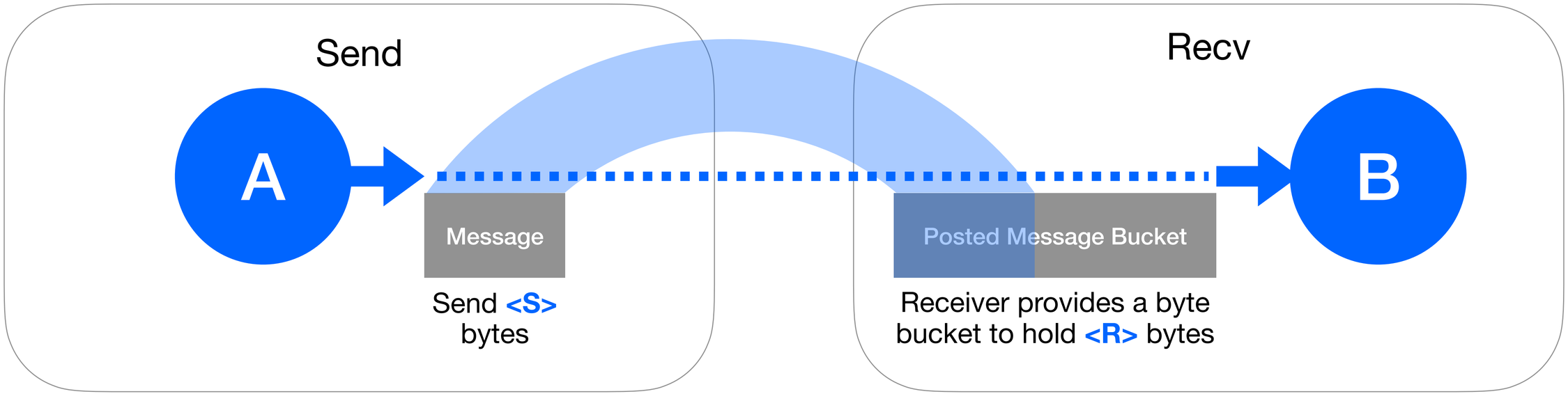
- **Every byte matters**
- Can support very large messages
- E.g. Distributed computation

Unreliable



- **Messages may drop, come out of order, or be duplicated**
- Unicast and multicast
- Usually small messages (i.e. packets)
- E.g. audio, video, lidar, radar

Two-Sided (Send/Recv) Semantics: Message Size



sizeof(<S>) <= sizeof(<R>)

I.e. sent messages can be smaller than what takyon's recv request provided as a byte bucket

Two-Sided (Send/Recv) Semantics: Message Order

Rule 1: The order of arriving messages is based on one of:

- If the Provider is ‘reliable’ then the messages will arrive in the same order as sent
- If the Provider is ‘unreliable’ then the messages may:
  - Arrive in a different order than sent
  - Be dropped and lost forever
  - Be a duplicate of a previously arrived message

Rule 2: The arrived message is put into the recv buffer based on:

- If takyonPostRecvs() IS supported: the order of received messages is defined by the order of posting recv requests
- If takyonPostRecvs() NOT supported: the order of received messages is defined by the order of calling takyonIsRecved()

Blocking versus Non-Blocking Transfers

Interconnects that can support non-blocking usually have a DMA engine

Blocking
CPU does transfer

takyonSend() - Send message and block until message is sent. Memory buffer can be updated when this call is complete.

takyonIsRecved() - Wait for a message to arrive. Memory buffer won't be overwritten until reused by subsequent call this this function.

takyonOneSided() - Transfer message and block until complete. Memory buffer can be updated when this call is complete.

Non-Blocking
DMA does transfer in the background

takyonSend() - start two-sided transfer

takyonIsSent() - block until sent

Don't modify message at this point

If not called before message arrives, message will be dropped (unreliable), or cause a failure (reliable)

takyonPostRecvs() - provides a place to recv data ahead of time

takyonIsRecved() - block until message arrives

Process the message before re-posting to avoid overwriting data from a newly arriving message

takyonOneSided() - start one-sided transfer

takyonIsOneSidedDone() - block until transferred

Don't modify message at this point

Non-Blocking Notifications and Fences

Send and one-sided completion notifications

- The application can decided if a completion notification should be used or not:

```
TakyonSendRequest.use_is_sent_notification = false;      // Don't call takyonIsSent()
TakyonOneSidedRequest.use_is_done_notification = false;  // Don't call takyonIsOneSidedDone()
```

- Since transfers are processed in order, can used a final transfer to see of if previous un-signaled transfer is completed. Improves latency and throughput.

Fences

- Forces preceding non-blocking transfers (send, read, write, atomics), where notification is turned off, to complete before the new transfer starts

```
TakyonSendRequest.submit_fence = true;
TakyonOneSidedRequest.submit_fence = true;
```

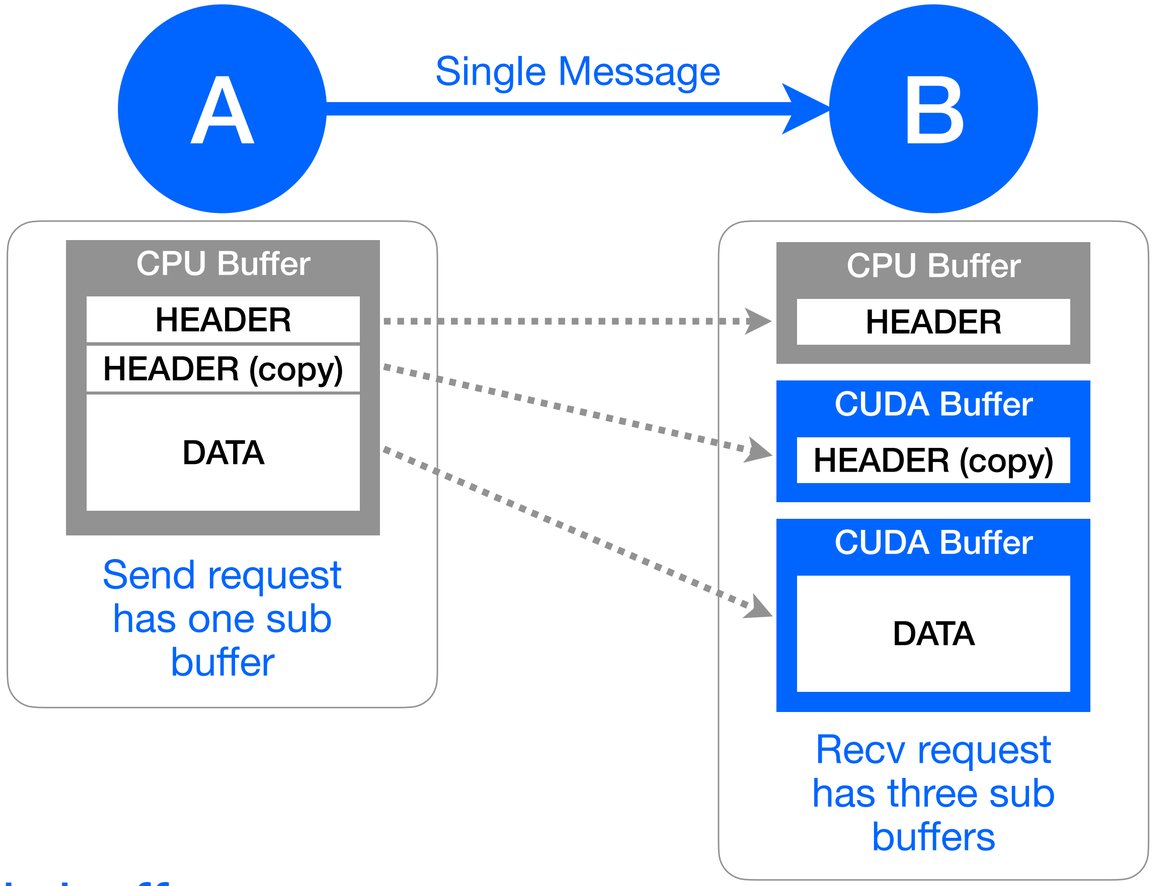
- This is typically only needed if a preceding 'read' or 'atomic' operation is invoked (changes local memory) just before sending the results of the preceding operations

Single Message, Multiple Sub Buffers

Multiple sub buffers may allow for highly organized and optimized processing

Hypothetical Example:

- It's common for GPUs to do heavy processing and CPU does light book keeping, but both need to know the attributes of the data



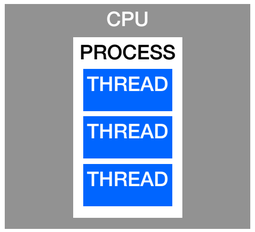
Some providers only allow one sub buffer per message

Fault Tolerant Communication

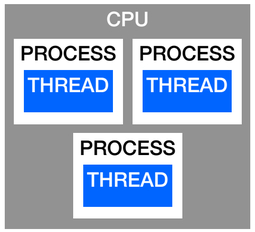
- Detecting degraded communication
  - Disconnect detected (e.g. network is down); i.e. Takyon function return false
  - Timeout (the transfer is not occurring in a reasonable amount of time)
- Handle a degraded communication path
  - Used dynamic path destruction/creation, without effecting other existing paths
  - Some other application defined alternative
- Notes about an app (not the communication API) being fault tolerant
  - The communication API should provided the hooks for fault tolerance
  - Only the app can know what to do when communication degrades
  - Communication paths should be independent (Want to avoid “One light goes out they all go out”)

Takyon is not fault tolerant (by design), but does provide fault tolerant hooks

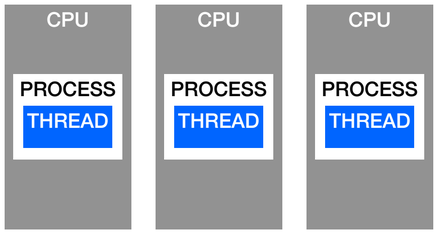
Accelerate Development by Locality Staging



1. Start with one process and multiple threads
   - While dataflow is being developed, only need to run a single executable
   - Easier to debug crashes or validate memory leaks/overwrites (e.g. valgrind)



2. Move to multiple processes on one CPU
   - Simple way to validate the migration of dataflow, without jumping to multi-processors



3. Move to multiple processors
   - Migration should be simple
   - Can now test for deployment performance

Most of the Takyon examples support this

Looking to the Future

Posible Enhancements

- Strided Transfers
  - Currently avoiding this since most interconnects don’t support this
- Publish/Subscribe
  - A potential replacement for the overly complex DDS
  - Could have simplified participants, publishers, subscribers, and QoS
  - Make messages opaque and private (removes need for DDS’s intermediate language)
- Collectives: barrier, scatter, gather, all-to-all, reduce, etc.
  - Already done as a separate API with Takyon 1.x, and may be converted to Takyon 2.x
  - Create a complimenting GUI to build and maintain the collective groups visually

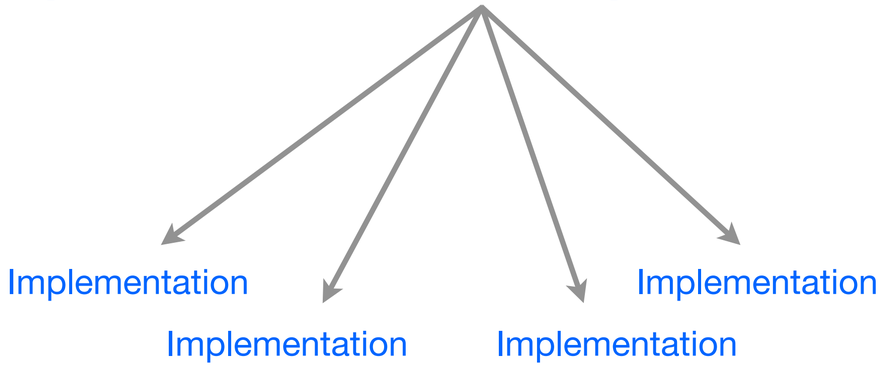
CHALLENGE: Is Takyon missing a key feature?

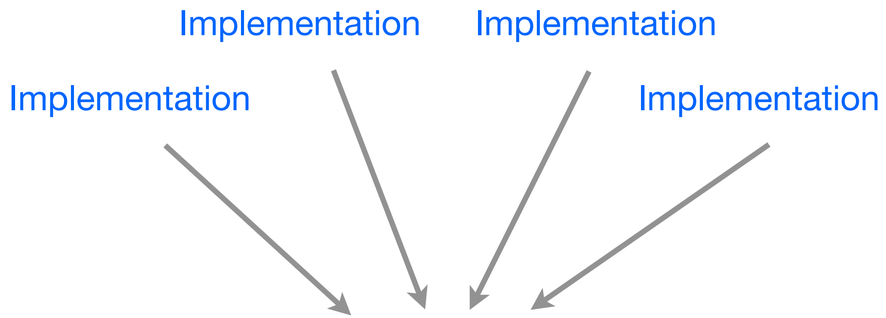
From Open Source to Open Standard

Takyon is looking to become an Open Standard

- Open standards make technology pervasive
- Open standards with rigorous conformance testing enable consistency across multiple implementations that can meet the needs of diverse markets, price points, and use cases
- Open standards often use open source to spread the implementation effort for sample implementations, tools, samples, conformance tests, validators etc.

Open Standard = Shared **Specification**





Open Source = Shared **Implementation**

Ultimate Goal: A Khronos Open Standard

Proven processes for creating royalty-free interoperability standards

KHRONOS GROUP

CONNECTING SOFTWARE TO SILICON

3DCommerce, ANARI, Camera, COLLADA, EGL, glTF, KTX, NNEF, OpenCL, OpenGL, OpenGL ES, OpenGL SC, OpenVG, OpenVX, OpenXR, SPIR, SYCL, Vulkan, Vulkan SC, WebGL, ML MACHINE LEARNING FORUM, SYCL SAFETY CRITICAL EXPLORATORY FORUM, KHRONOS SAFETY CRITICAL ADVISORY FORUM SC

Khronos uses a methodical “New Initiative” process to evaluate industry interest in new standardization proposals. This process makes it straightforward for any company that perceives an industry need to propose a new collaborative activity at Khronos—whether or not they are Khronos members.

Learn more at: https://www.khronos.org/exploratory/

Heterogeneous Communication proposal is here, ready to form a new exploratory group

Proposal
Heterogeneous Communication API

Exploratory Group
Determines industry interest and create detailed Statement of Work

SOW

Working Group
Meet to work on detailed design work. Open to Khronos Members

Khronos Standard

High-level discussions, requirements, design directions and design contributions

Takyon = A design contribution for a Heterogeneous Communication API

Call to Action

Get involved to help shape Heterogeneous Communication Standards

- **Call for participation in a Khronos Exploratory Group**
- Open to all at no cost
- Goal of the group is to explore industry interest in the creation of open royalty-free API standard for Heterogeneous Communication
- All participants will be able to discuss use cases and requirements for new interoperability standards to accelerate market growth and reduce development costs for Heterogeneous Communication
- Design contributions will be considered and Takyon is currently the only design contribution, but others are welcome
- If the Exploratory Group reaches significant consensus and industry support then Khronos will work to initiate a Working Group to start the detailed work of defining an industry standard

Khronos Exploratory Group Enquiries: marketing@khronosgroup.org

Takyon Questions: michael23.both@gmail.com