SLIPSTREAM: AUTOMATIC INTERPROCESS COMMUNICATION OPTIMIZATION

ILLINOIS

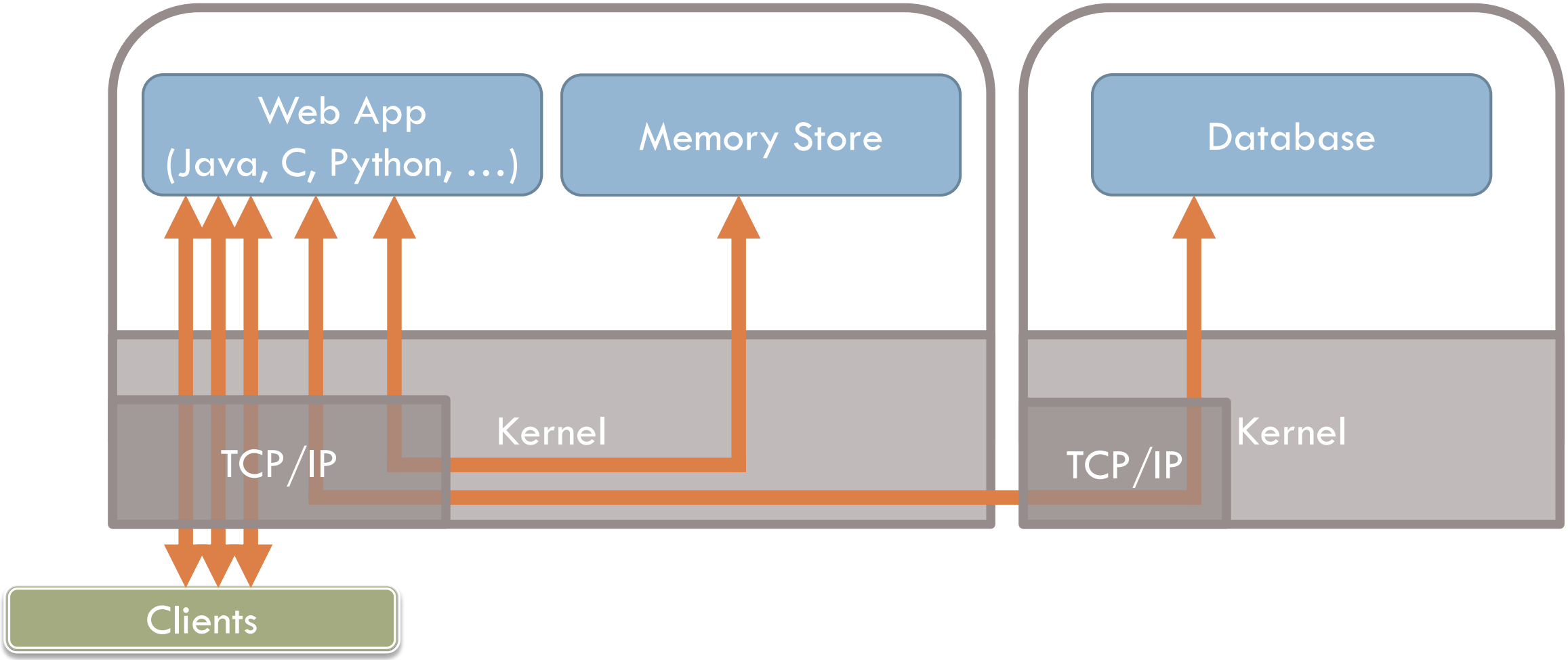
Will Dietz, Joshua Cranmer, Nathan Dautenhahn, Vikram Adve

Introduction

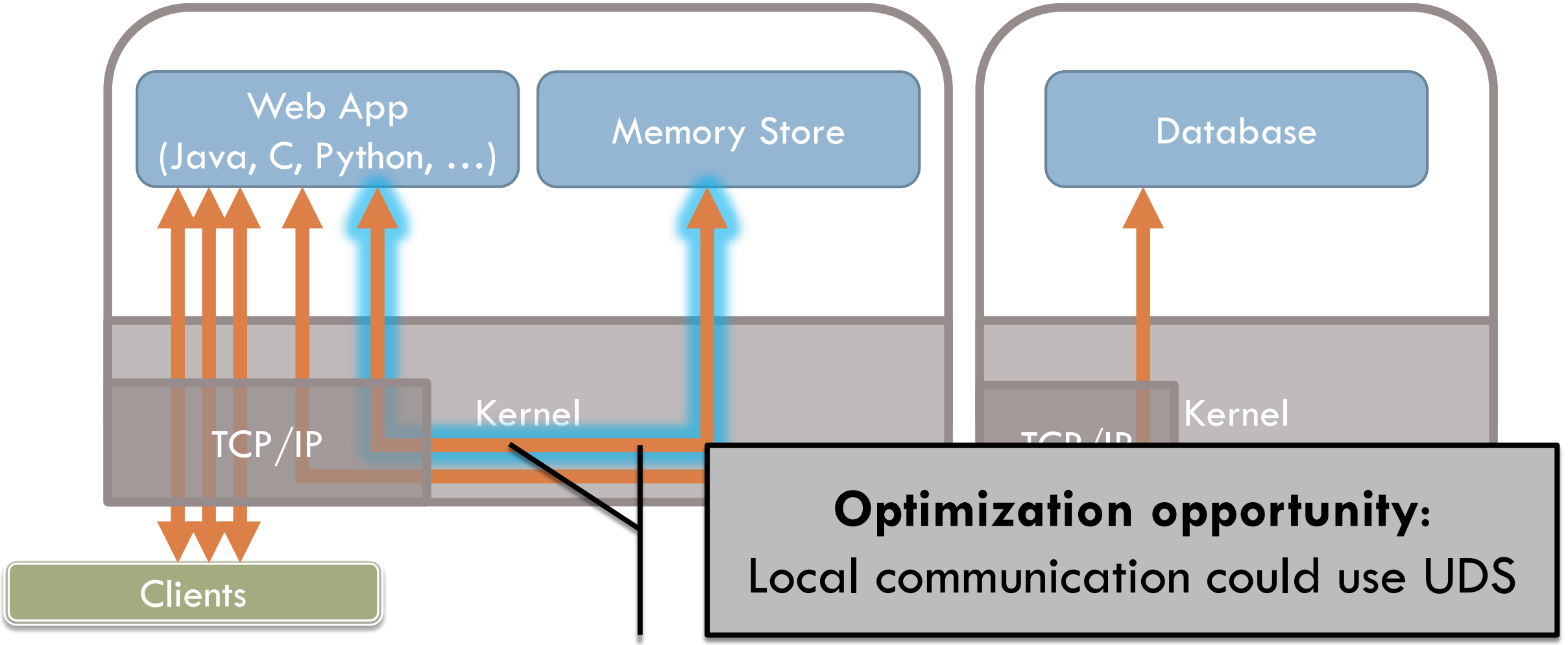
- **Use of TCP is ubiquitous**
  - Widely Supported
  - Location Transparency
- **Programmer-friendly but not always ideal**
- **Faster IPC exists for local communication**

- **Goal: Best of TCP generality with fast performance locally**

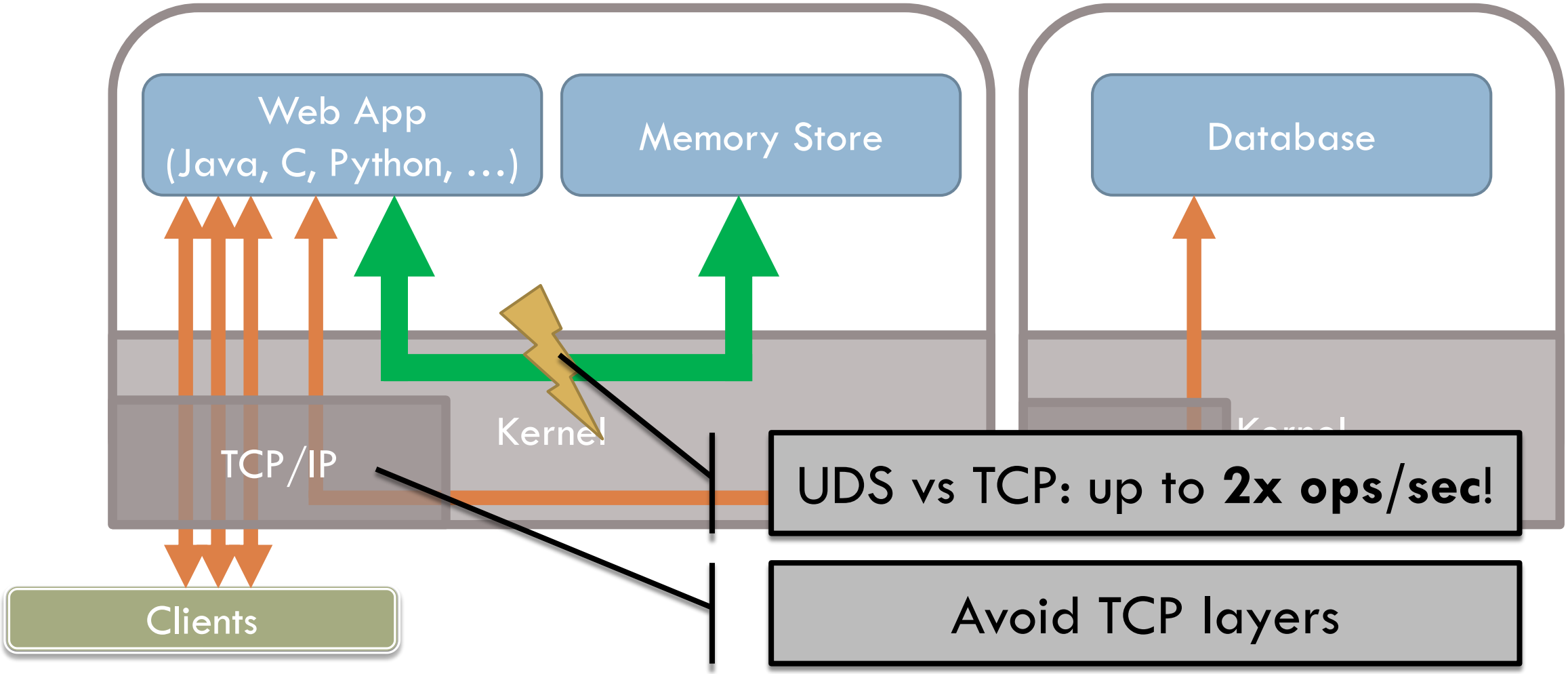
Motivating Example



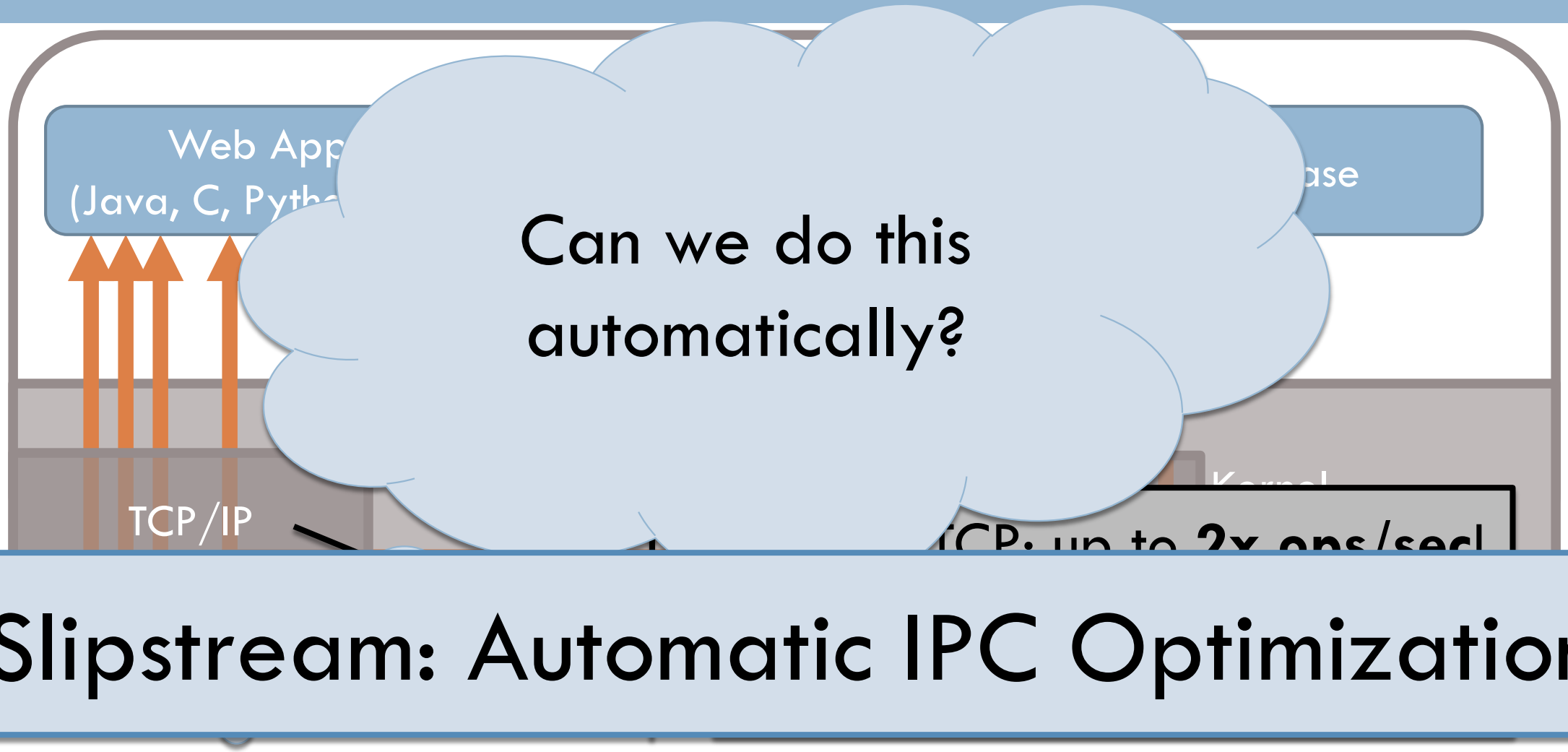
Motivating Example



Motivating Example – Manually Optimized



Motivating Example – Manually Optimized



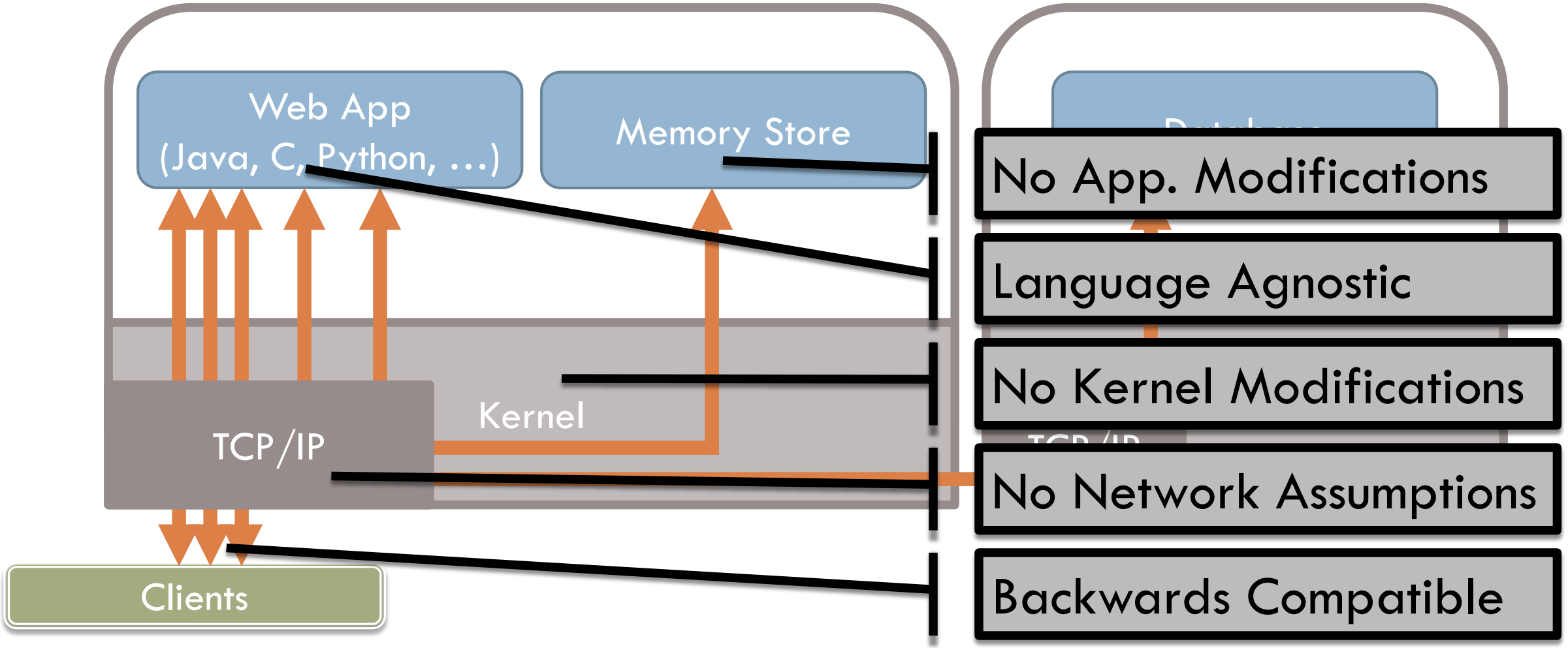
Slipstream: Automatic IPC Optimization

What's Coming

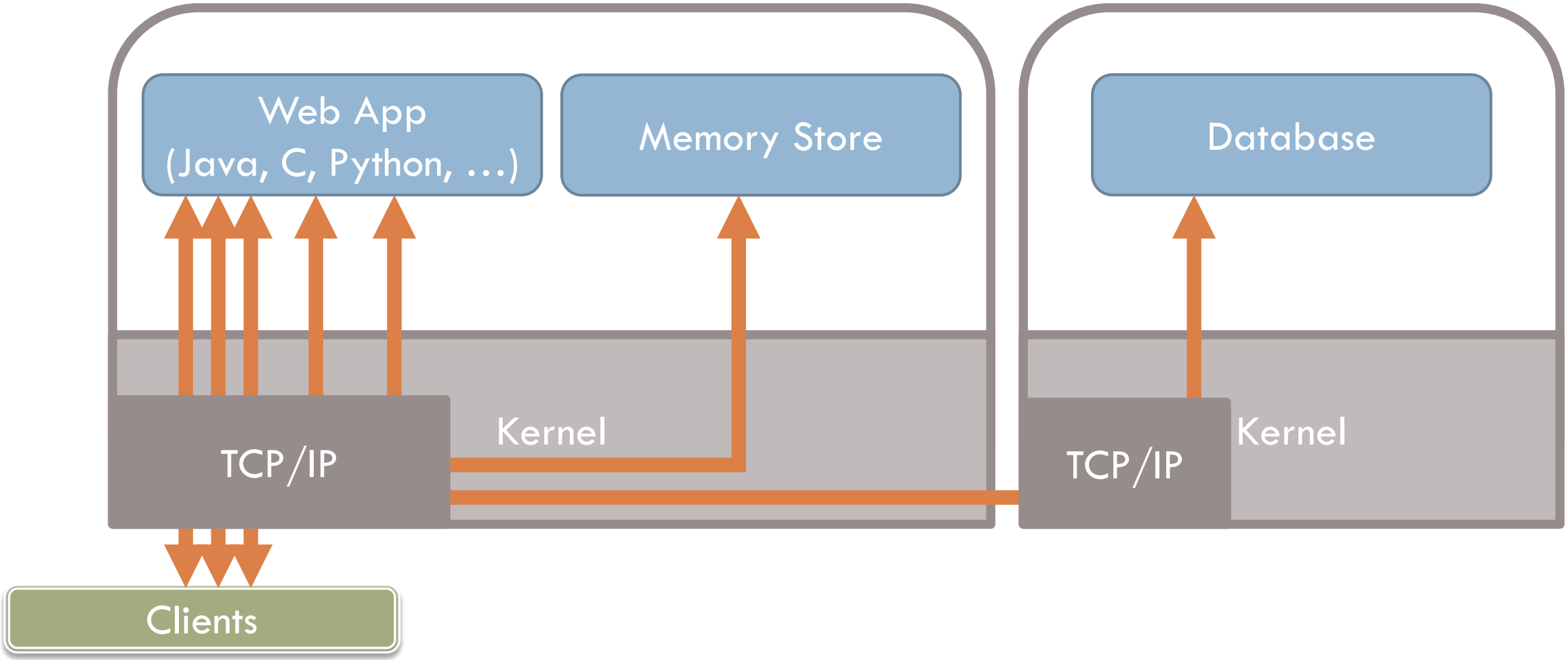
- Slipstream Overview
- How Slipstream Works
- Selected Results
  - Performance
  - Compatibility
  - Docker
- Conclusions

Slipstream Overview

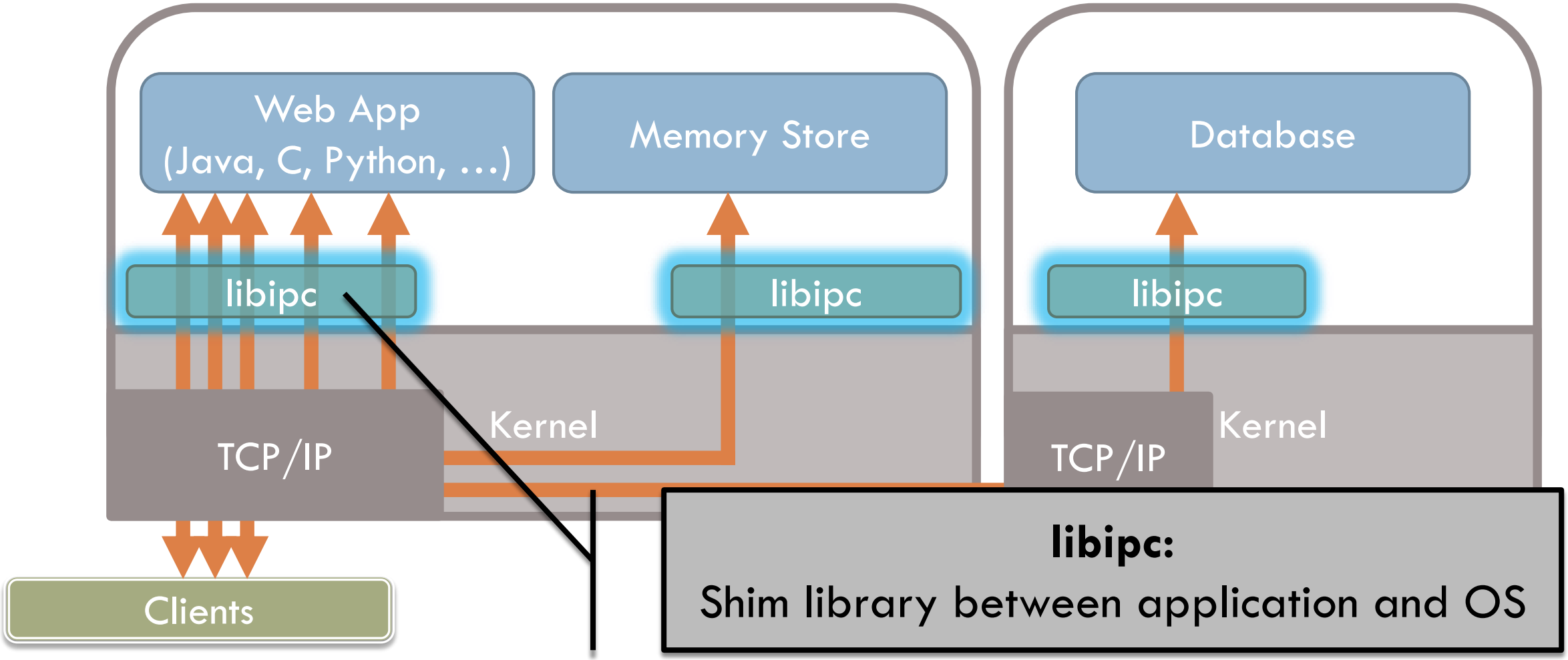
Design Principles: Automatic and Useful



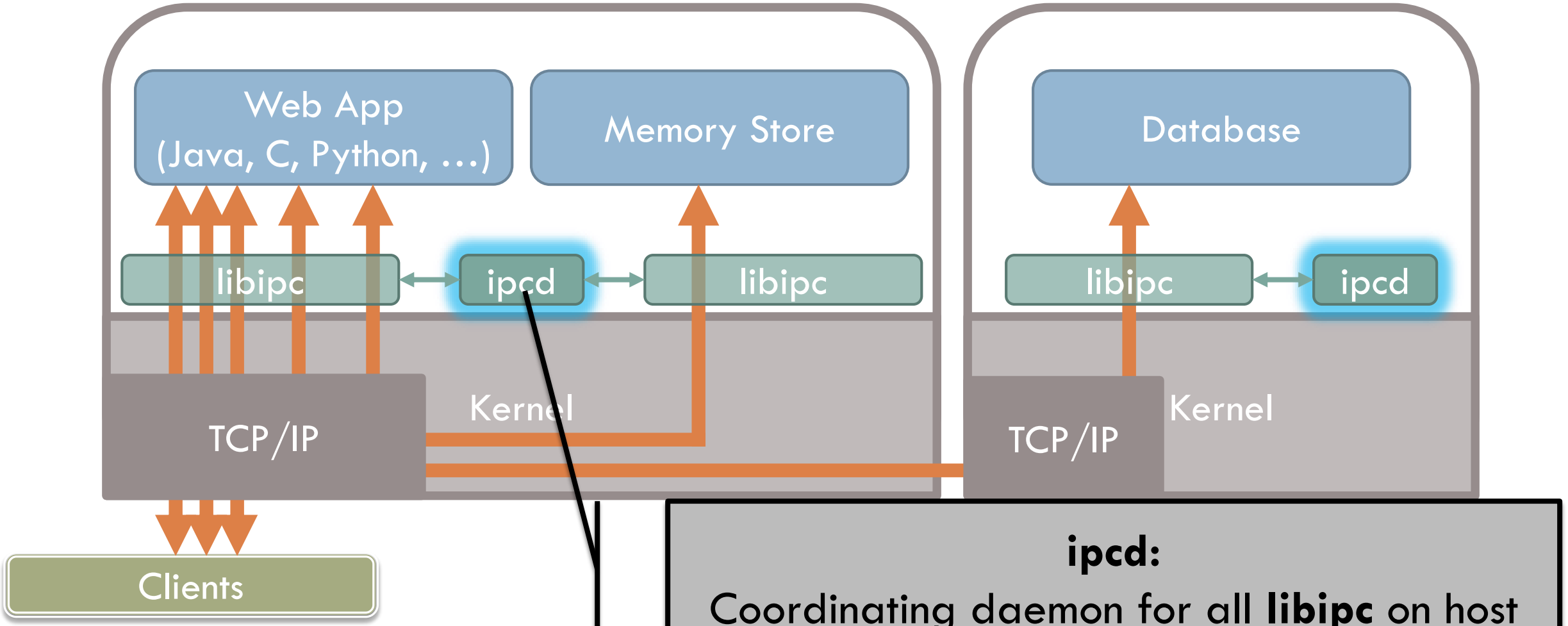
Slipstream Architecture: What is it?



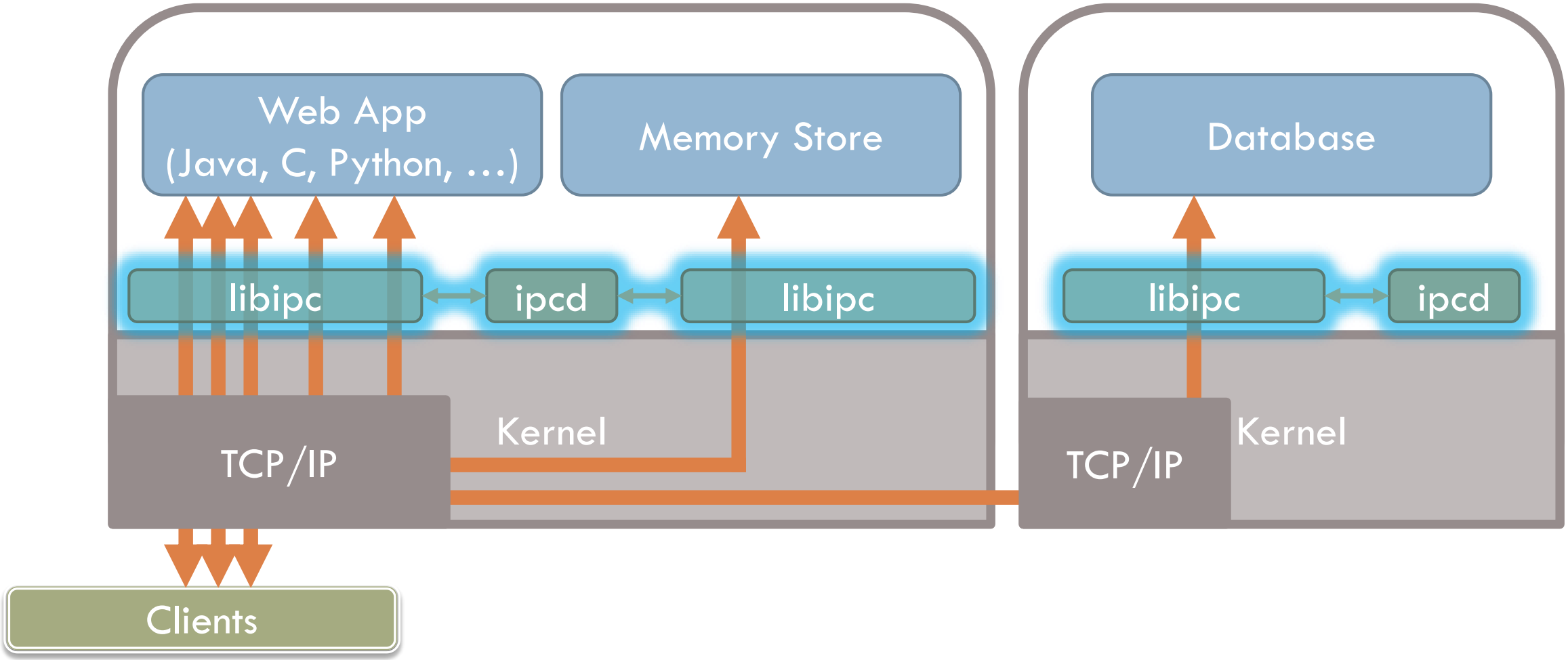
Example + (libipc)



Example + (libipc + ipcd)



Example + Slipstream

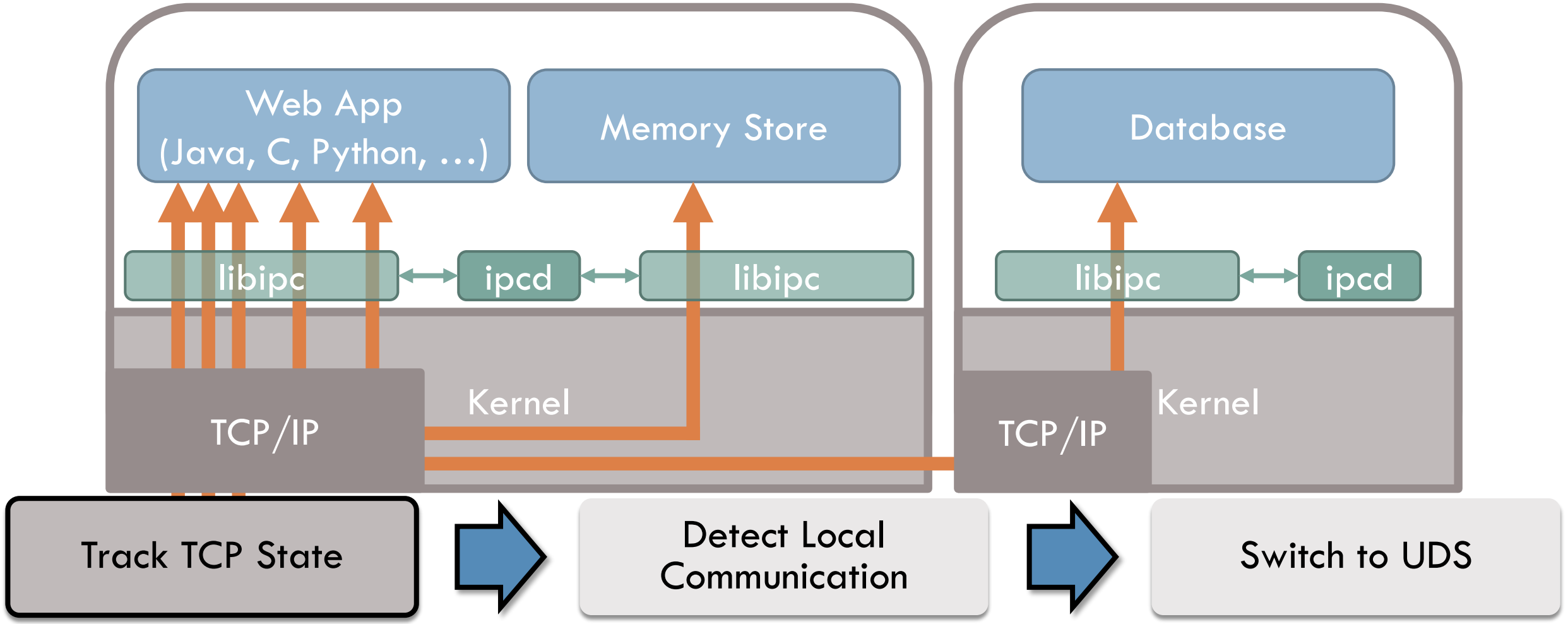


How Slipstream Works

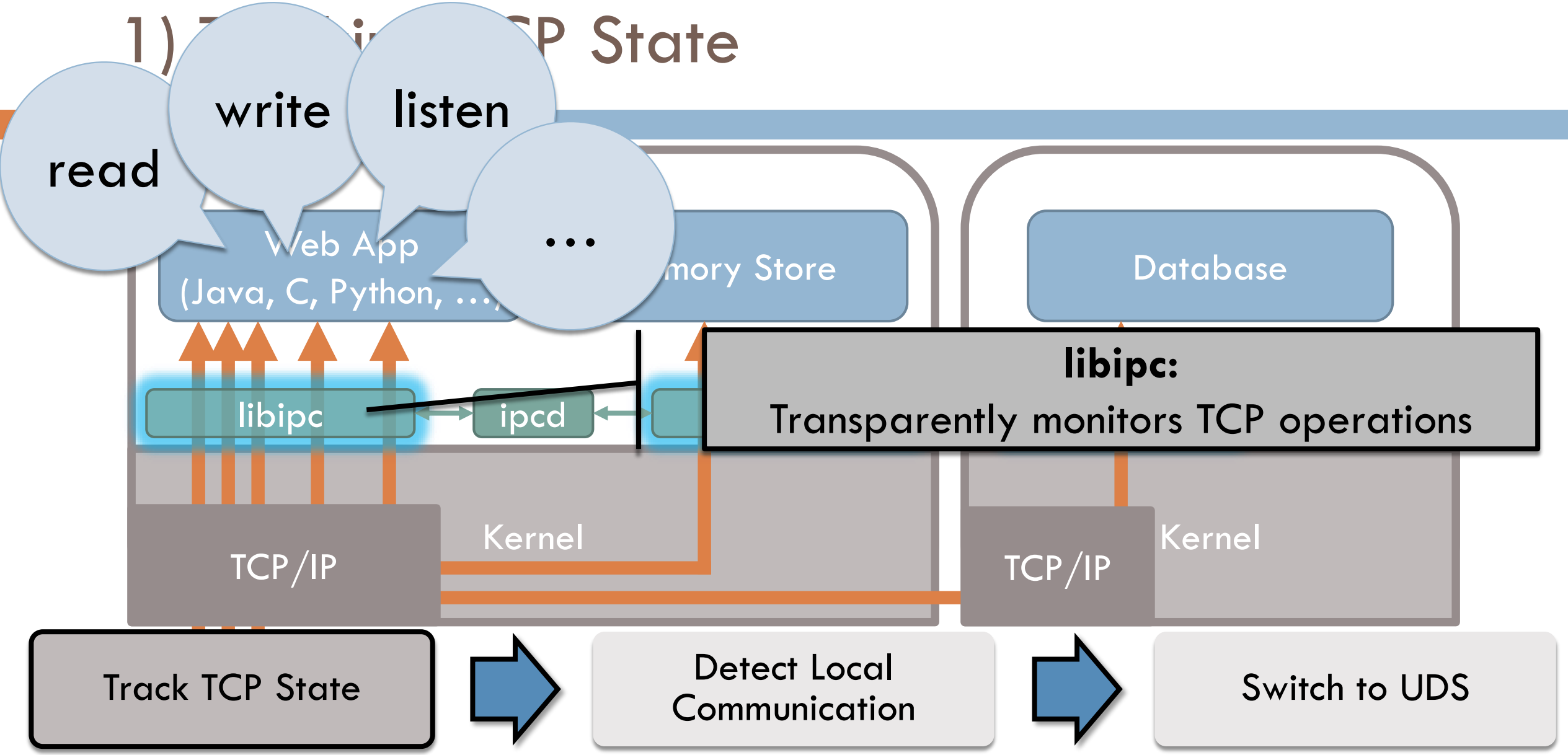
How Slipstream Works: Overview



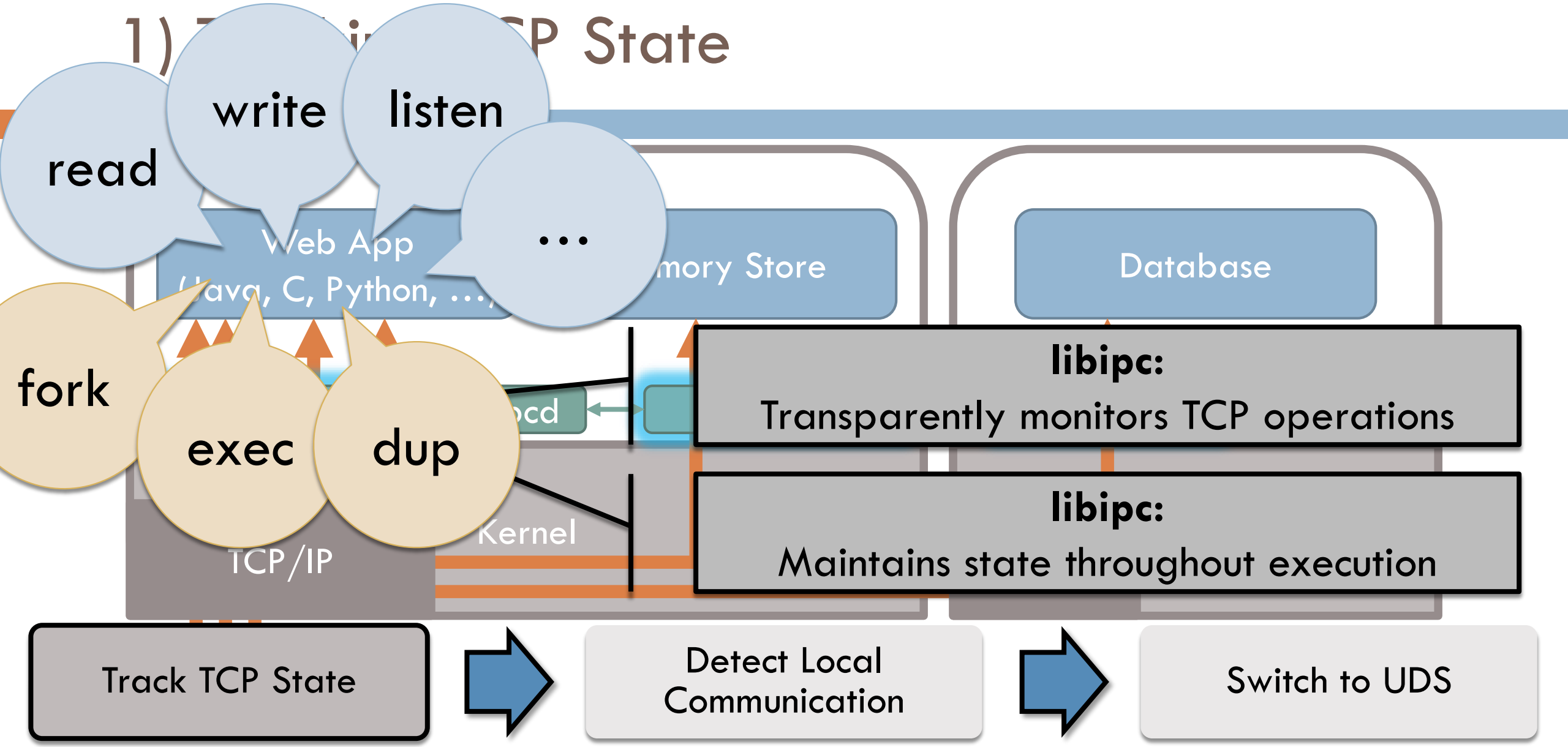
Tracking TCP State



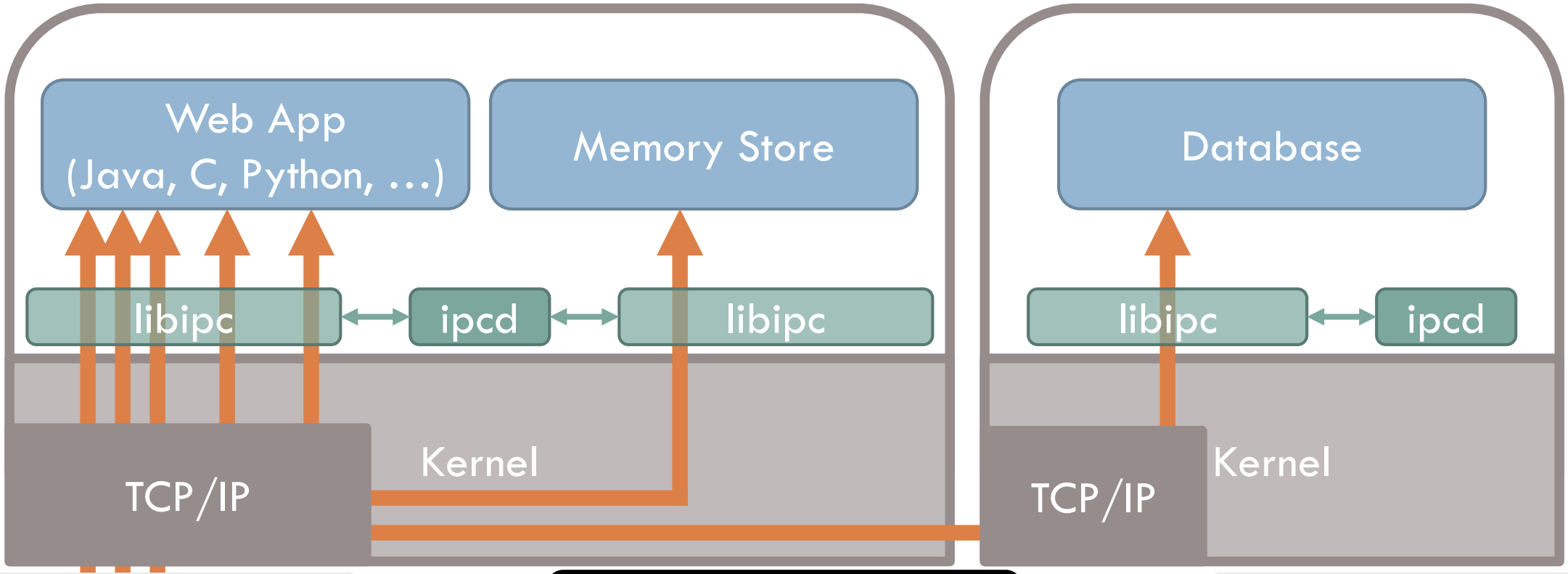
1) Tracking TCP State



1) T… i… CP State

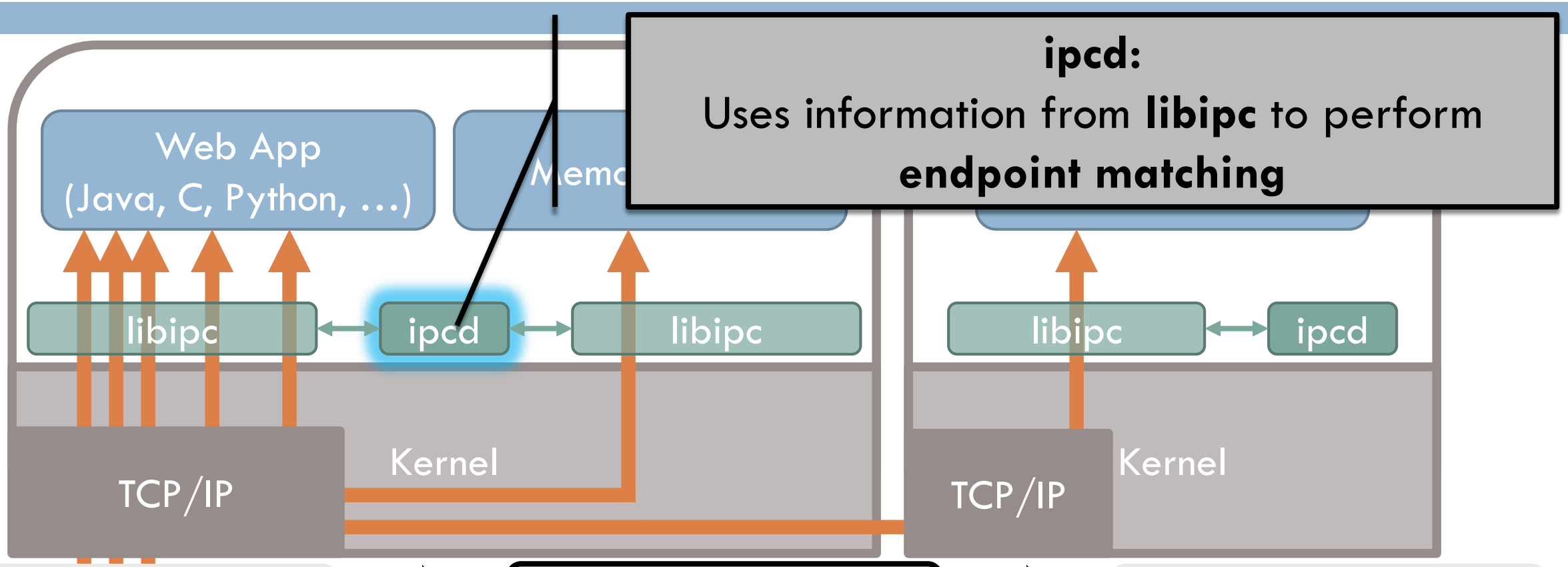

read
write
listen
…
fork
exec
dup
Web App
(Java, C, Python, …)
mory Store
Database
ocd
Kernel
TCP/IP
libipc:
Transparently monitors TCP operations
libipc:
Maintains state throughout execution
Track TCP State
Detect Local Communication
Switch to UDS

Detecting Local Communication



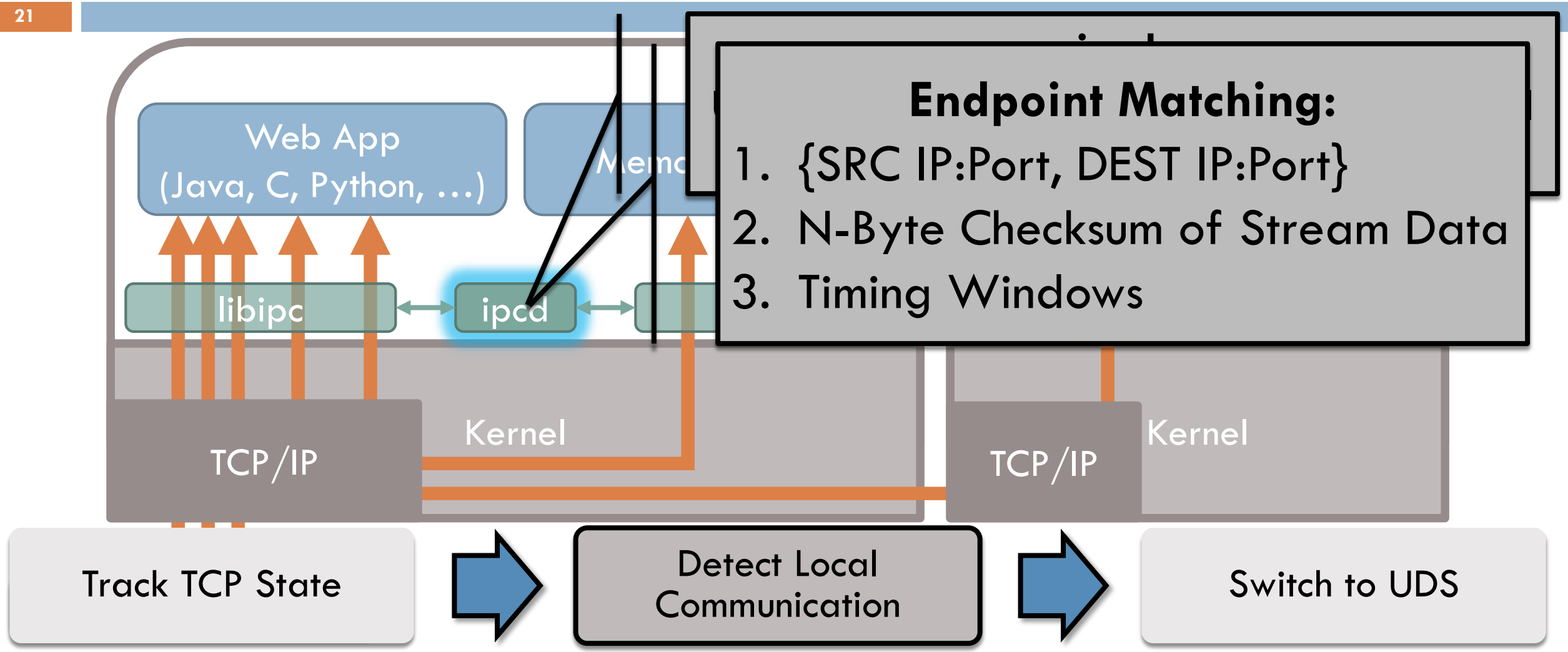
Track TCP State → Detect Local Communication → Switch to UDS

Detecting Local Communication

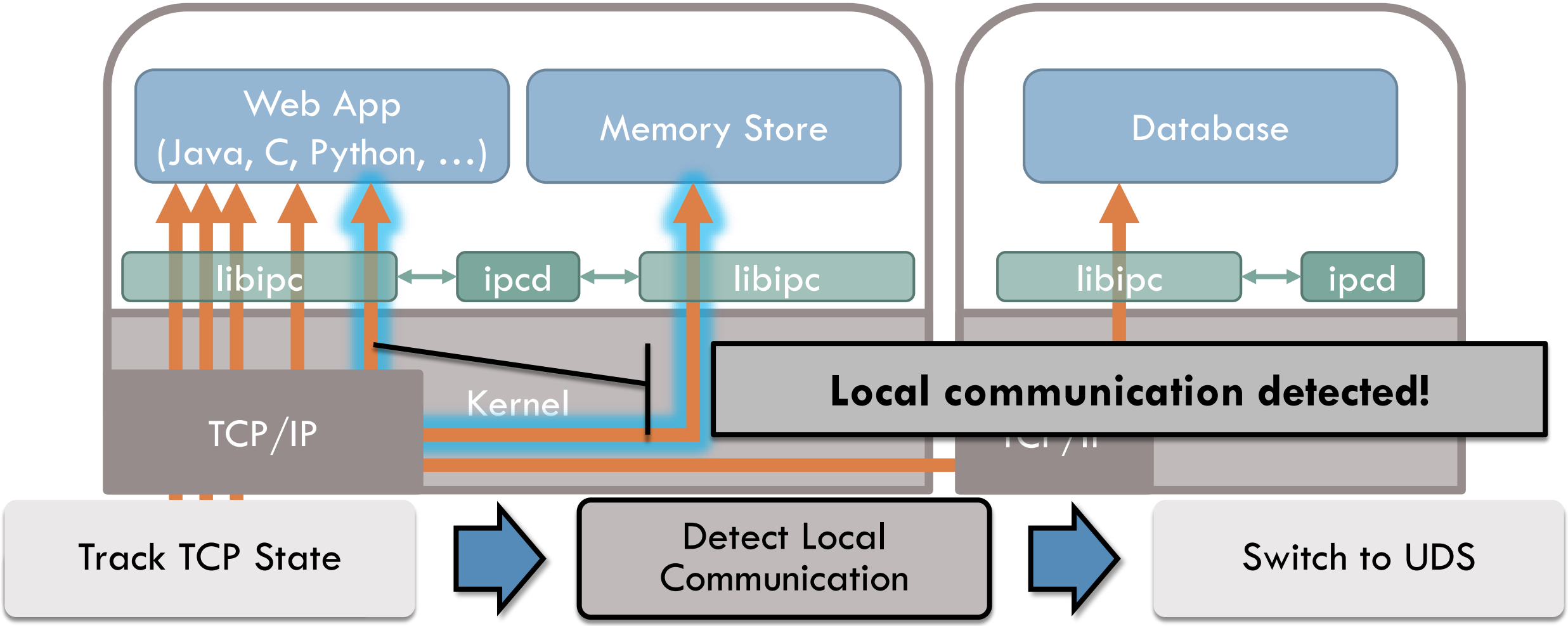


Track TCP State → Detect Local Communication → Switch to UDS

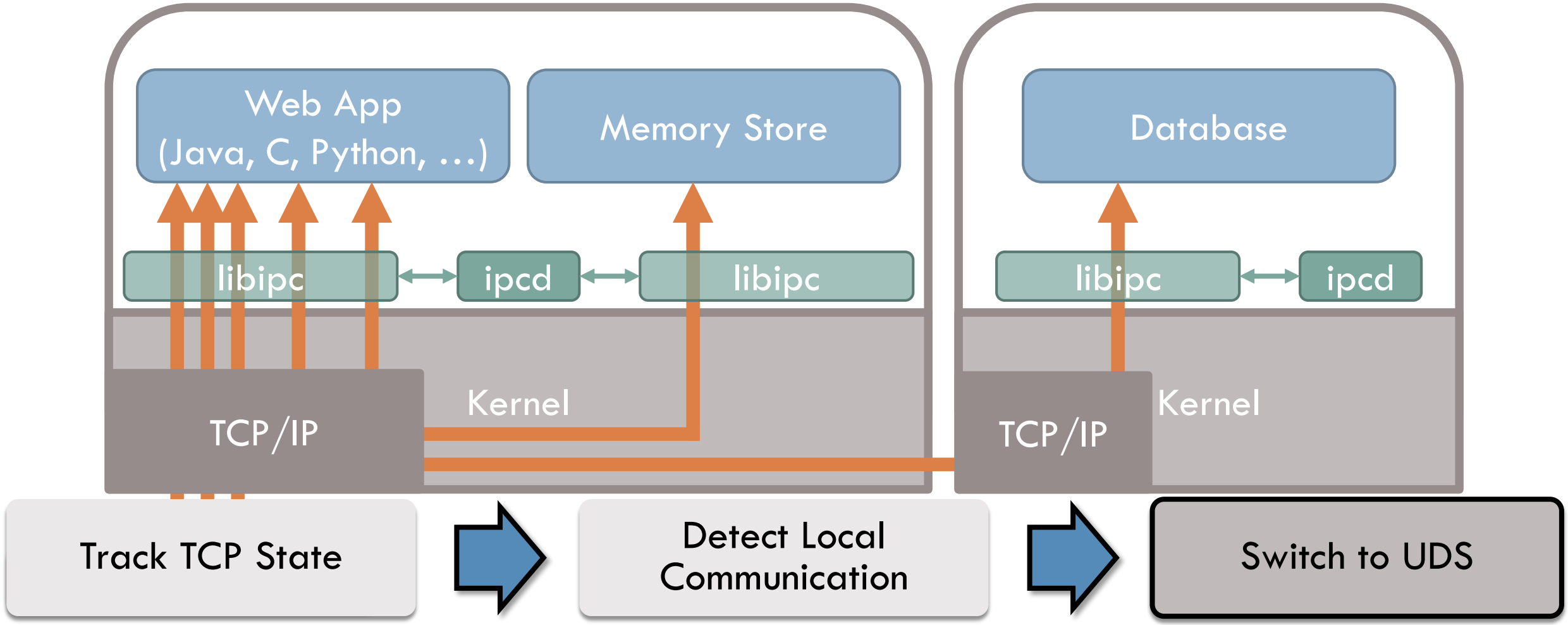
Detecting Local Communication



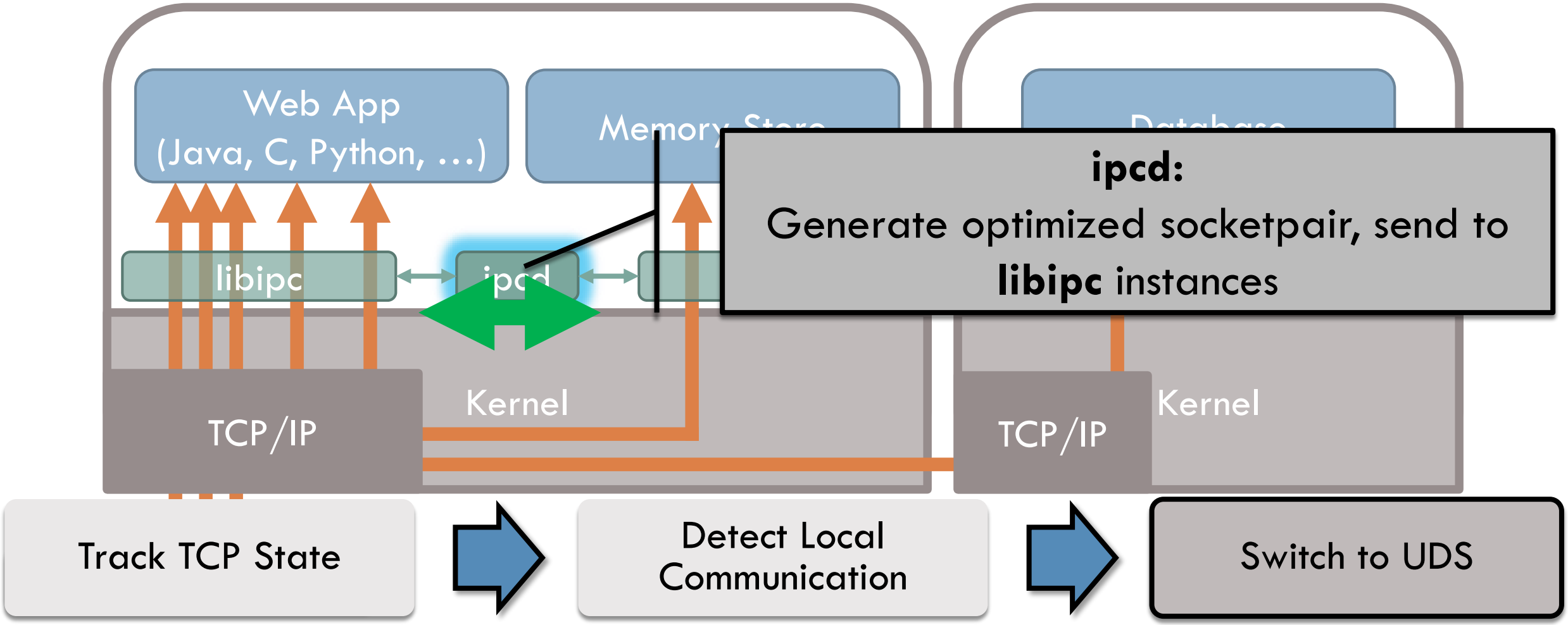
2) Detecting Local Communication



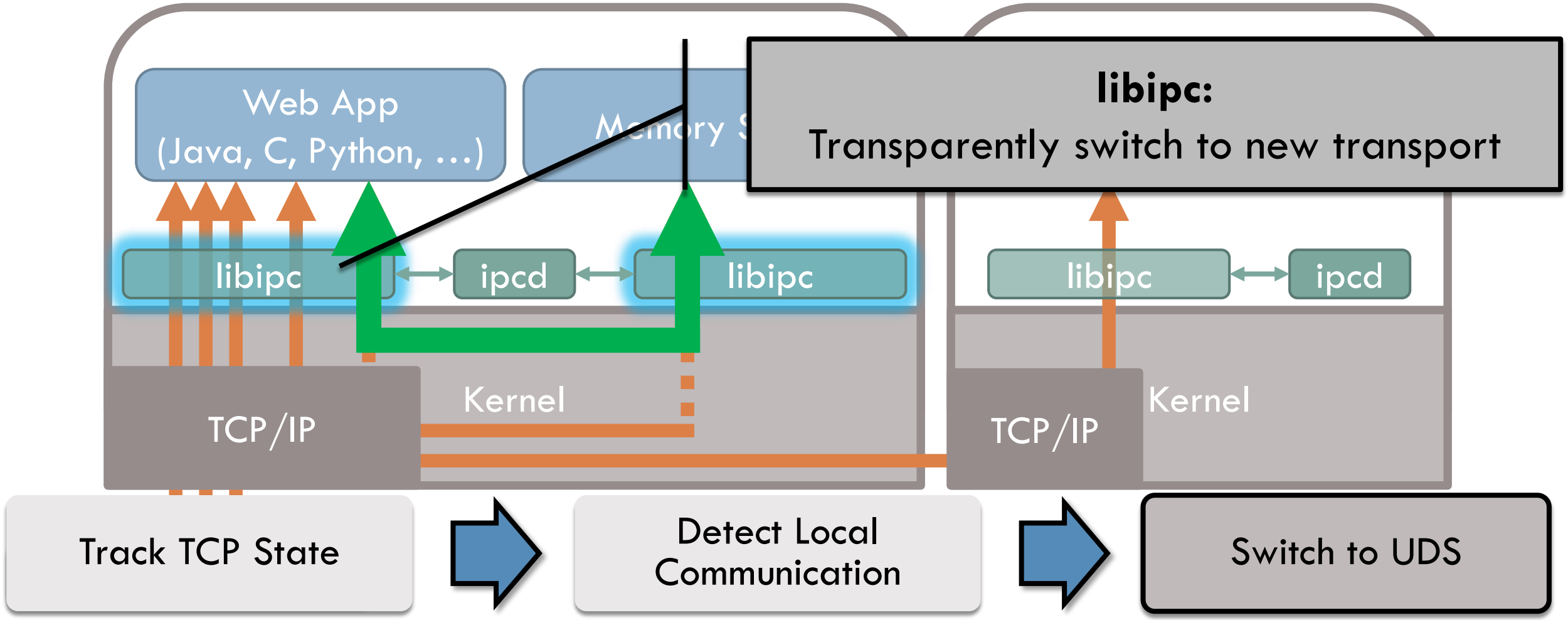
Switching to UDS



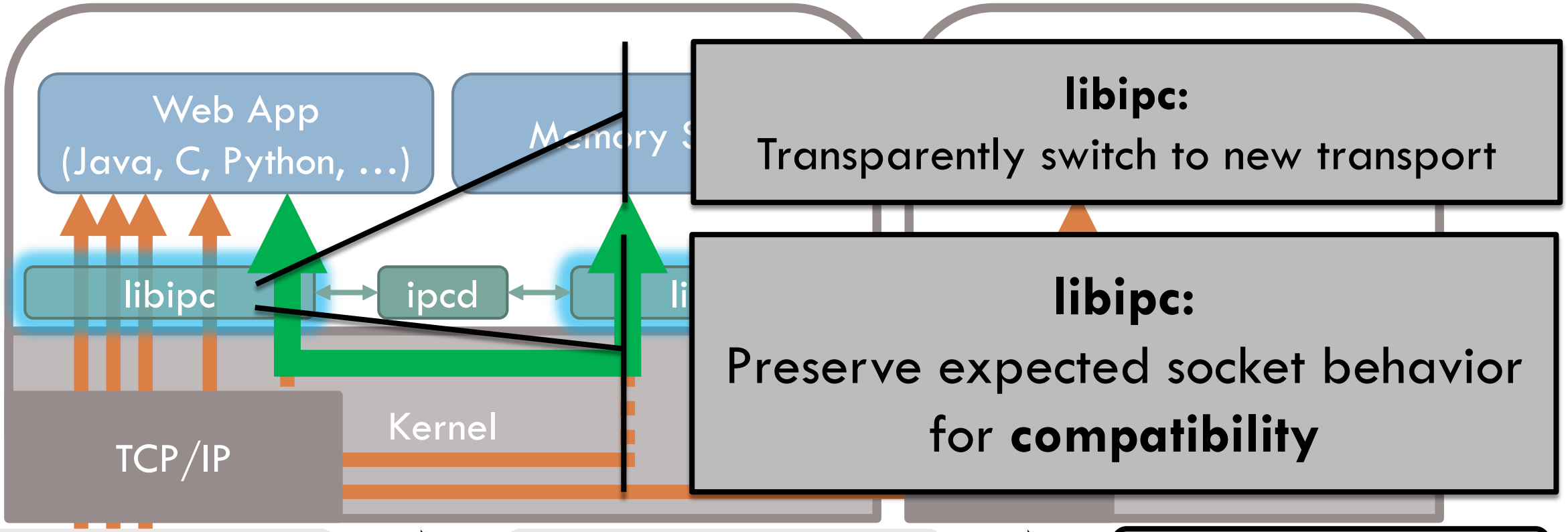
Switching to UDS



Switching to UDS

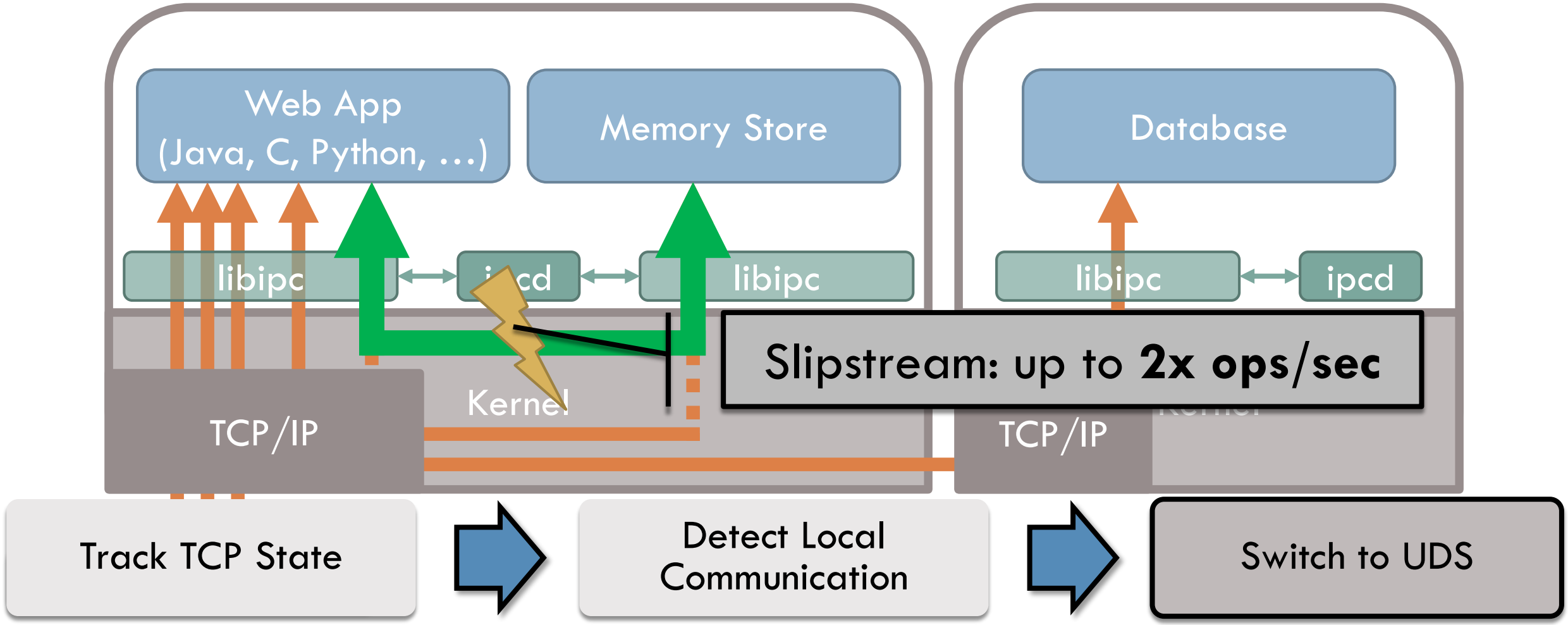


Switching to UDS



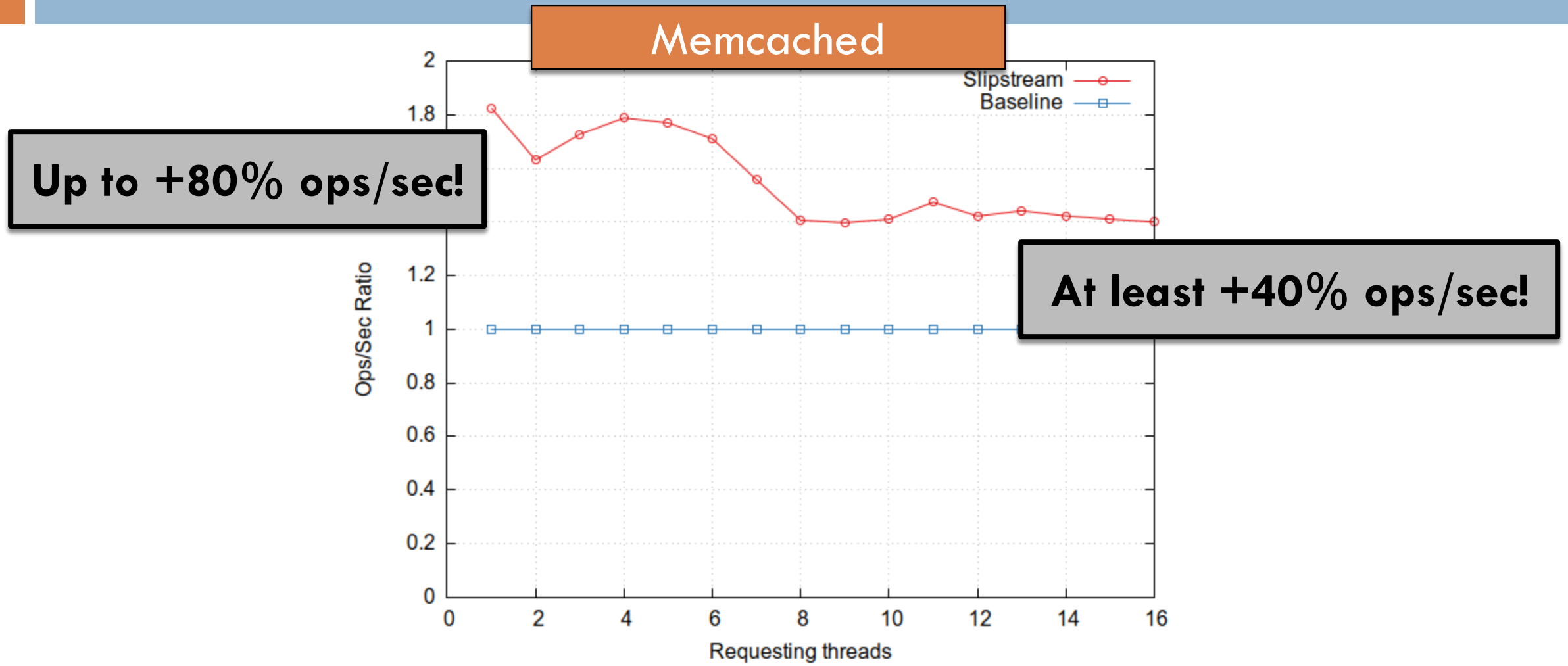
Track TCP State → Detect Local Communication → Switch to UDS

Switching to UDS

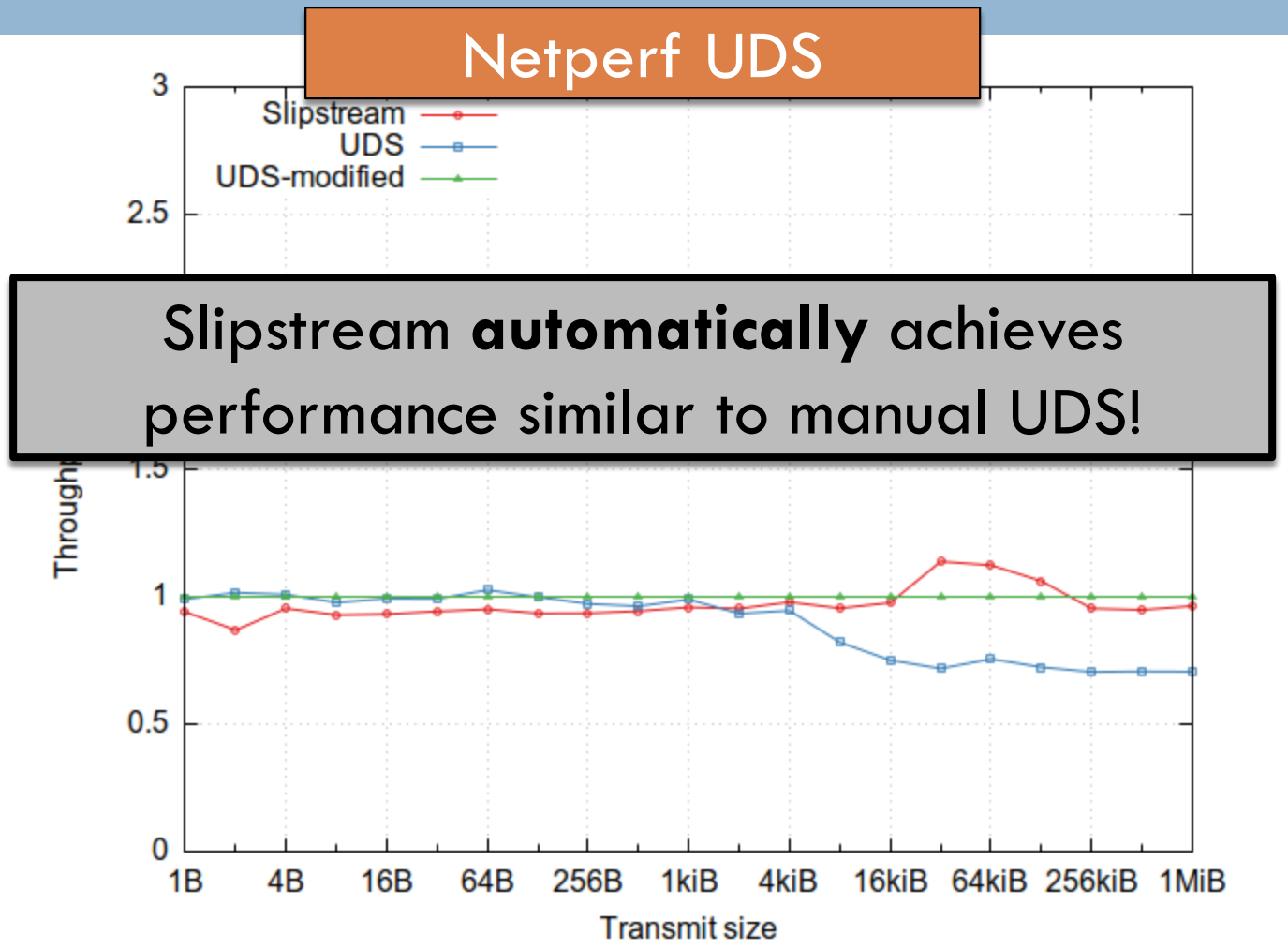


Selected Results

Performance vs Local TCP

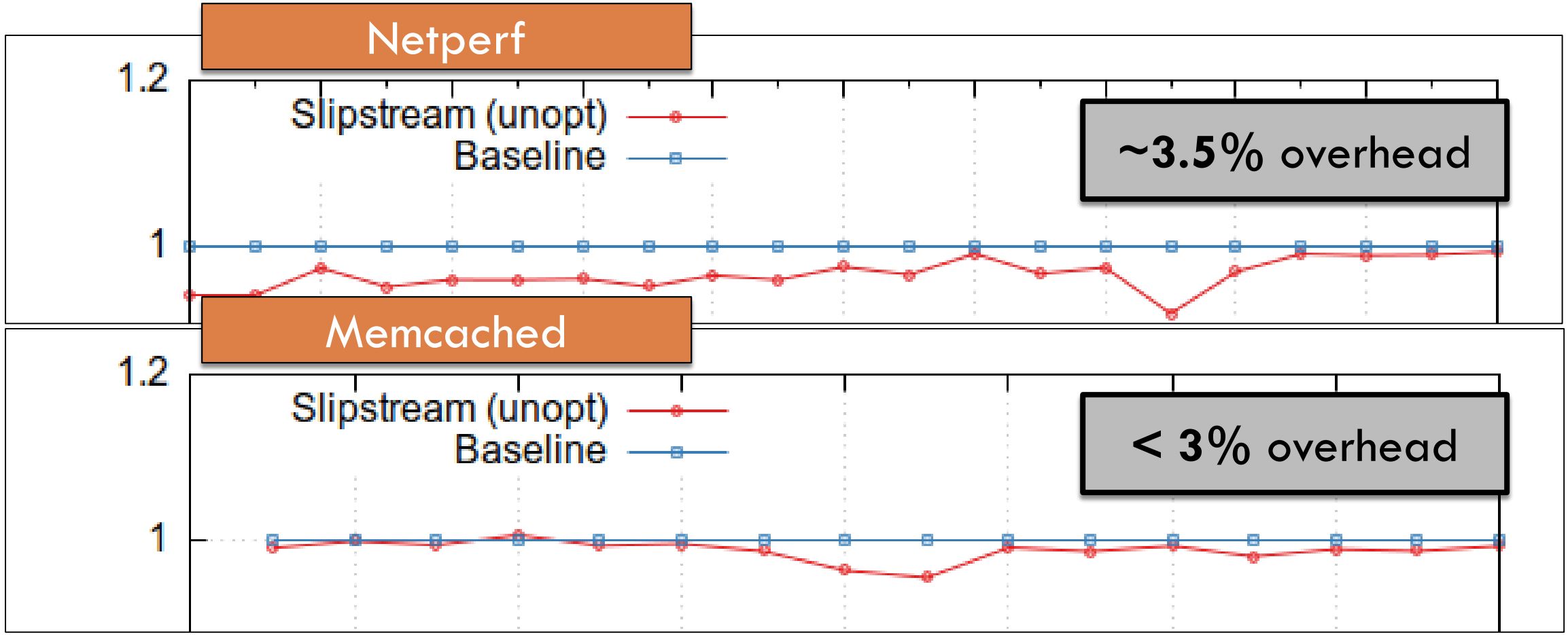


Performance vs Manual UDS



Slipstream **automatically** achieves performance similar to manual UDS!

Performance Overhead



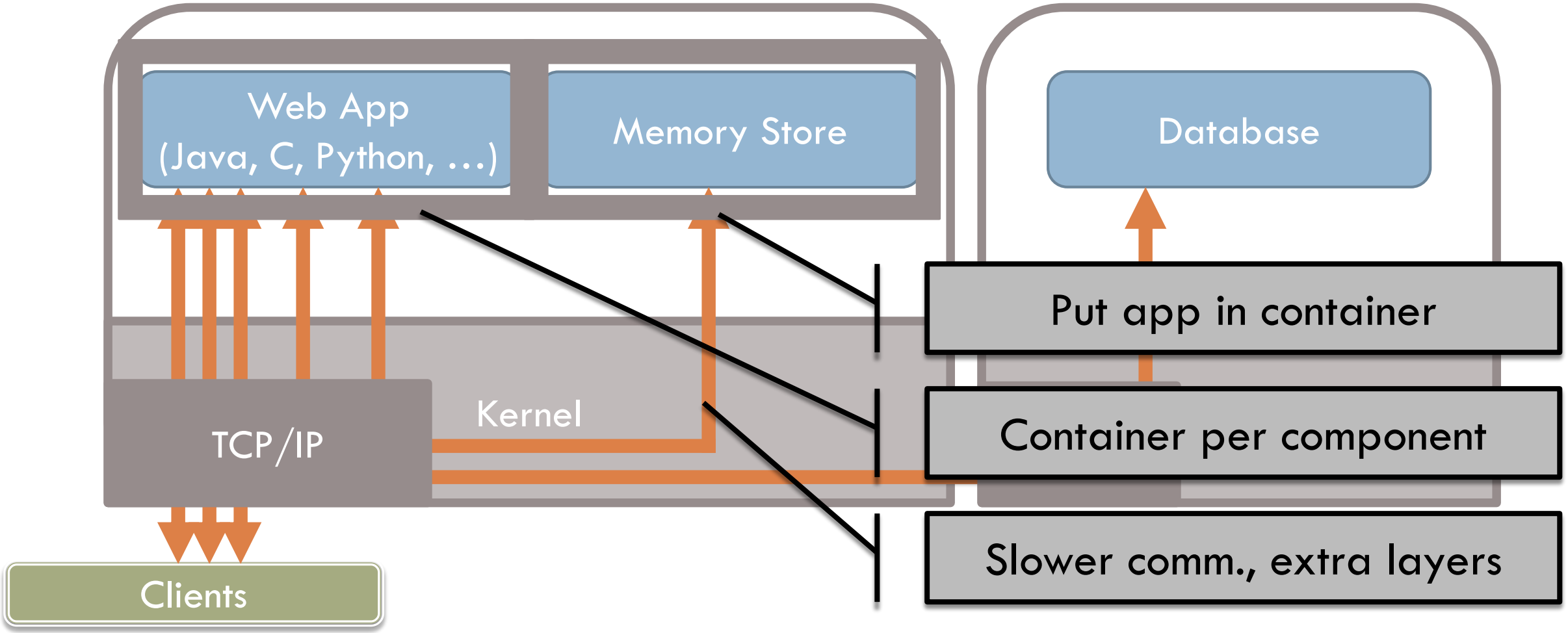
Software Compatibility

- **Supports many popular server applications**
  - MySQL, PostgreSQL, Redis, Memcached, Apache, Jenkins, …
- **Language agnostic** in practice
  - Java, C, Python, …

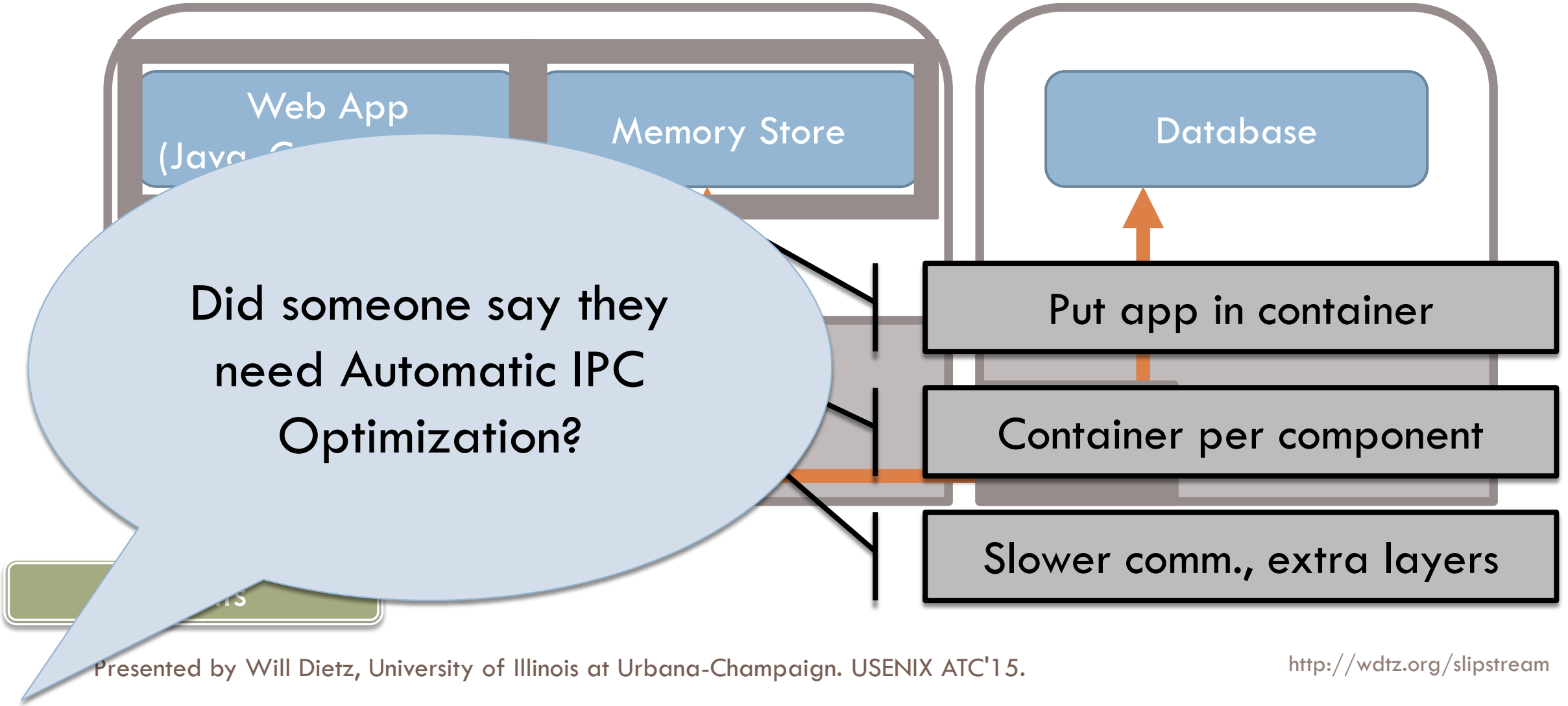
- **All work correctly with Slipstream**
- **All are optimized when communicating locally**

- **Supports use with Docker!**

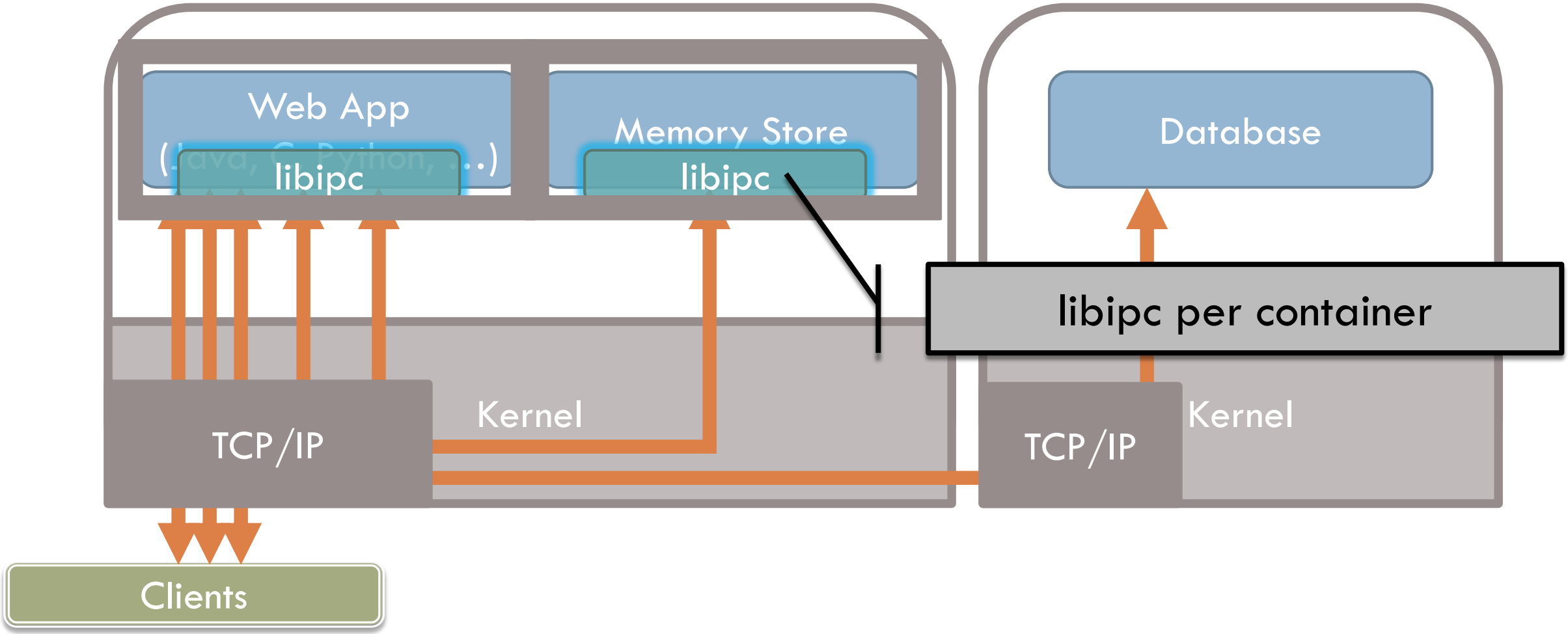
Example with Docker



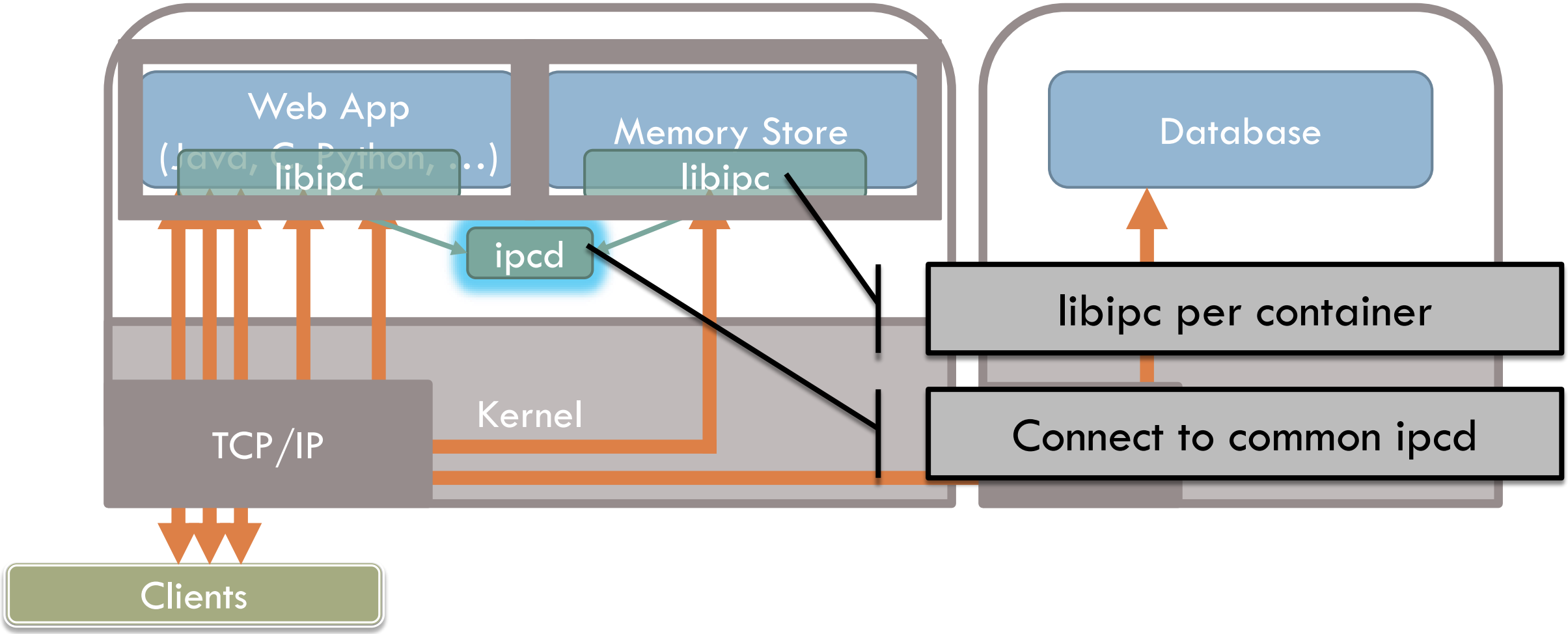
Example with Docker + Slipstream?



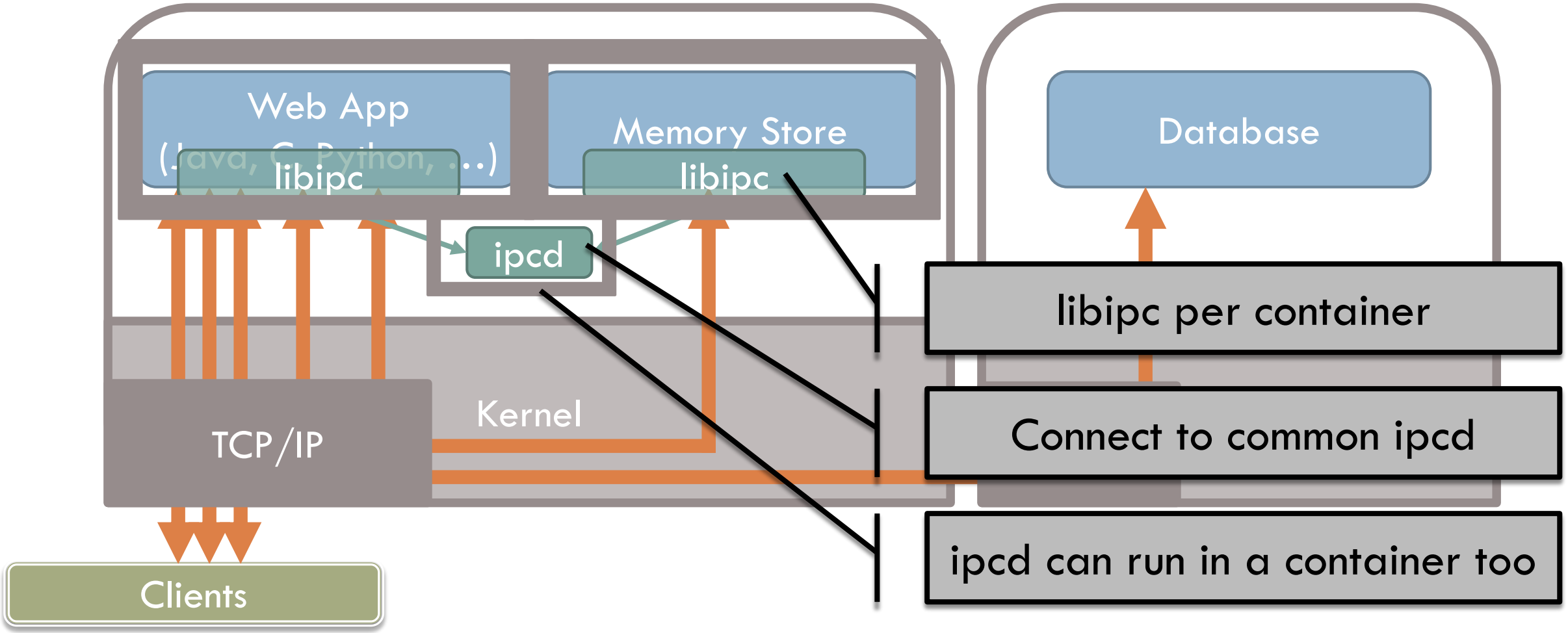
Slipstream with Docker



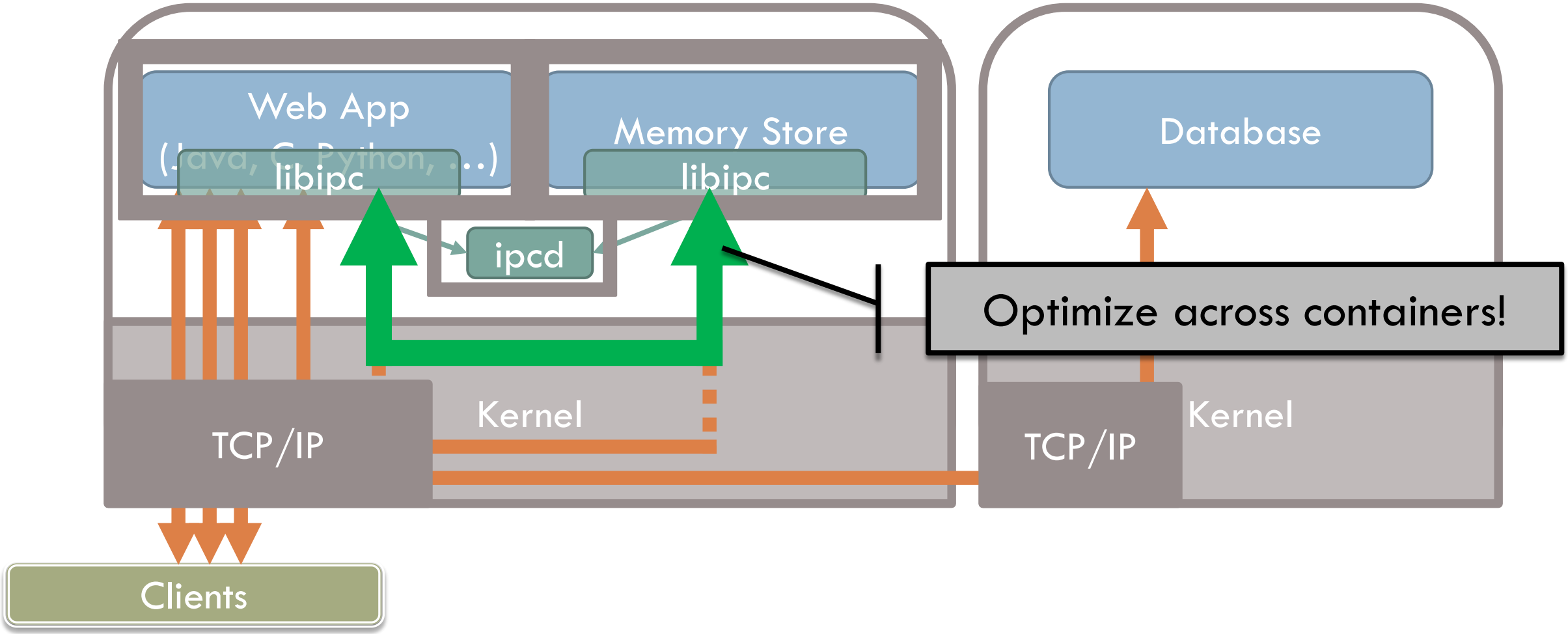
Slipstream with Docker



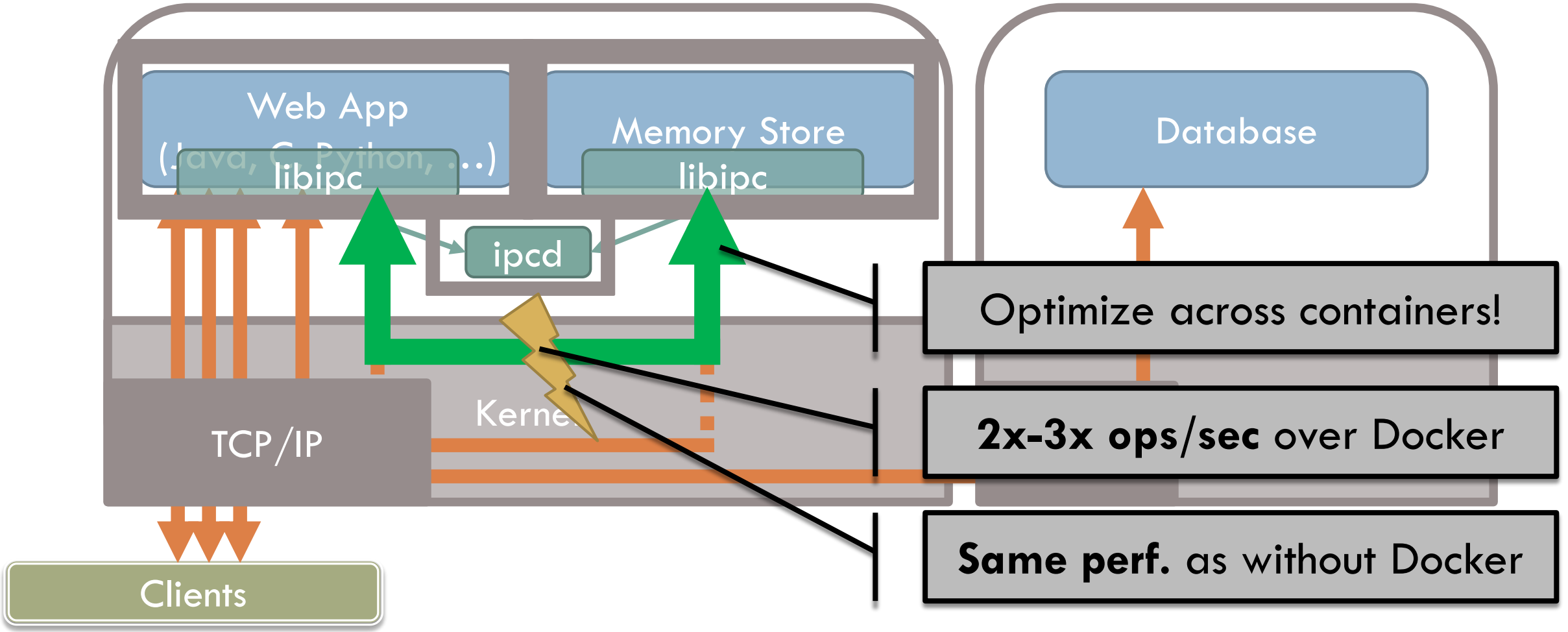
Slipstream with Docker



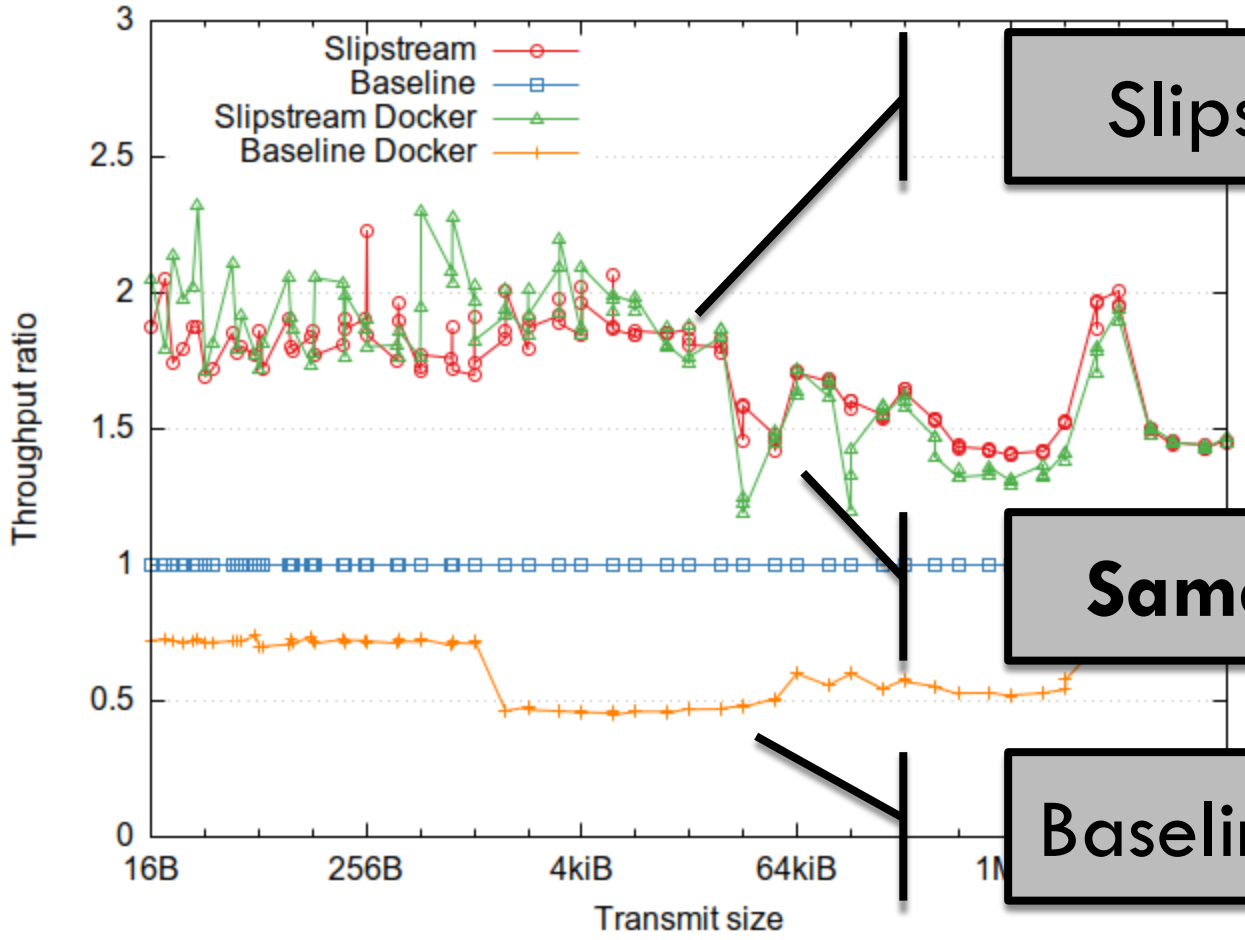
Slipstream with Docker



Slipstream with Docker



Docker with NetPIPE-C



Slipstream shows **large speedups**

Same perf. with or without Docker!

Baseline Docker is significantly slower

Conclusions:

- **Automatically** use faster IPC mechanism when available
- **Easy to deploy, highly compatible**
  - No OS modifications
  - No modifications to application
- **Language-agnostic**
- **Backwards-compatible** (partial deployment supported)
- **~2x bandwidth improvement** for host-local communication
- **Low overhead** when optimization is not possible
- **Freely Available** at http://wdtz.org/slipstream

Thank you! Questions?

(End of presentation)

FAQ

Q: Why use UDS instead of other local IPC?

- Simply: Socket interface of UDS facilitated prototyping
- Any local IPC mechanism should be possible
  - (Patches welcome! ☺)
- Others could **avoid kernel interaction** for even better performance
  - See ipc-bench paper
- **Slipstream is not UDS-specific**
  - UDS implementation detail, idea is faster performance leveraging locality
  - In fact, netperf sometimes slower with UDS (see paper).

Q: Why not have application do this itself?

- **Legacy:** many applications don't do this today
  - (Knowledge that UDS faster than TCP is not new!)
- **They shouldn't have to,** Slipstream! ☺
  - Achieves performance of manual optimization
  - Low overhead when optimization not possible
- **Applications have better things to do than transport selection!**

Q: Why userspace solution, not in-kernel?

- Mostly: **Legacy**
- Ease of use, ease of deployment
- Suitable for use on other systems (FreeBSD, etc.)
- May not be appropriate for kernel
  - Correctness concerns
    - (Opt-in approach likely appropriate)
- Possibility for using even more efficient transports than UDS
  - Can't cut out kernel with a kernel-based solution

Q: What about frequent short-lived connections?

- Current implementation not well-suited for this, but could be!
- Would not benefit from optimization
  - only applied to longer-lived connections
- In fact, initial connection latency over localhost increased
  - Not by design, result of implementation choices (but nevertheless)
- Probably better off with custom "is local" decision mechanism
  - Ex: "Is destination IP 127.0.0.1"
  - Would allow optimizing before connection, get faster connect() times
- Thanks for the question, Joshua ☺.

Extra Slides

Related Work

Category	Name	Application Transparency	OS Transparency
OS Feature	Windows Fast-Path		
	Solaris TCP Fusion	✓	
	AIX fastlo		
	Linux TCP Friends	?	
VM-VM	XWay	✓	
	XenSocket		
	XenLoop	✓	
Userspace Networking Stack	mTCP		
	Sandstorm		
Userspace Shim Library	Fable	✓	
	Java Fast Sockets	✓	✓
	Universal Fast Sockets	✓	✓
	FastSockets	✓	✓
	Slipstream	✓	✓

Building on Slipstream

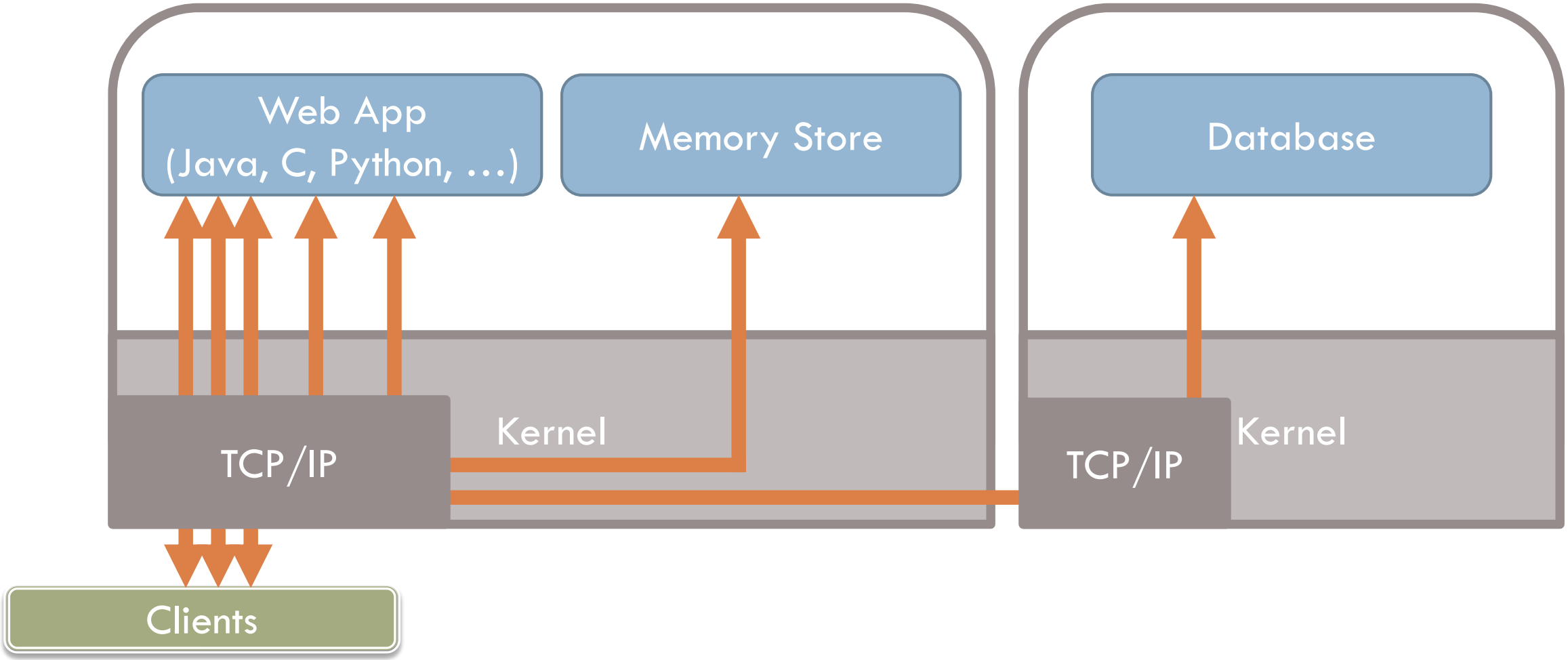
- Alternative transports (shared memory, etc.) (see ipc-bench)
  - Select best transport for workload/configuration
- Support multiple locality detection mechanisms
  - Provided one is suitable for many use-cases
  - If can assume network topology, can directly determine this
- More exhaustive testing, performance investigation

Try it out today!

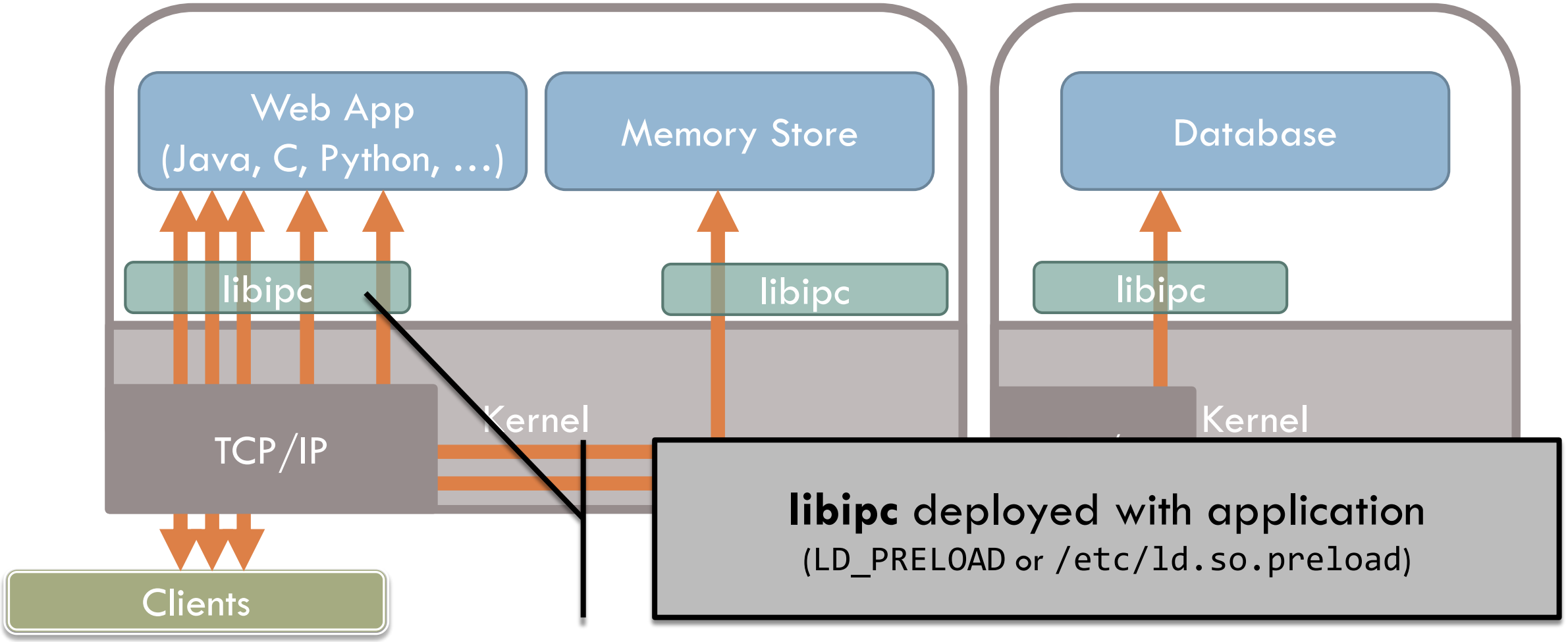
- No configuration, simple deployment
- Big performance gains for many applications (Docker support!)
- Low overheads when optimization is not possible
- Makes most of partial deployment
- Open-source, CRAPL (don't judge, it's research!)
- Download today!
  - http://wdtz.org/slipstream

Deploying Slipstream

Example + Slipstream: Deployment

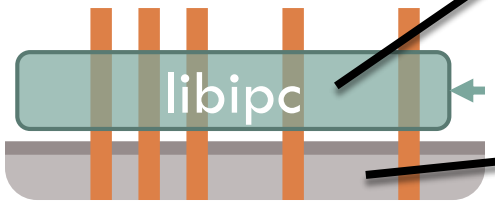


Example + Slipstream: Deployment



Challenges and Solutions

Example + Slipstream: Key Challenge

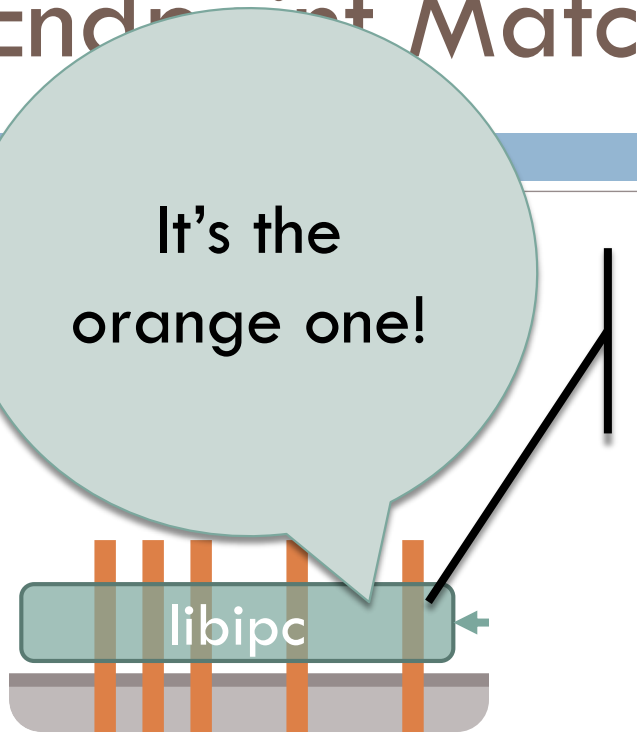


Challenge:
Which connections are communicating locally?

libipc only sees what goes through it

Solution:
Endpoint matching algorithm to identify **local communication**

Endpoint Matching Overview



libipc works with **ipcd** which has more information

Observed behavior from both **libipc** instances used by **ipcd** to find match

Endpoint Matching Overview

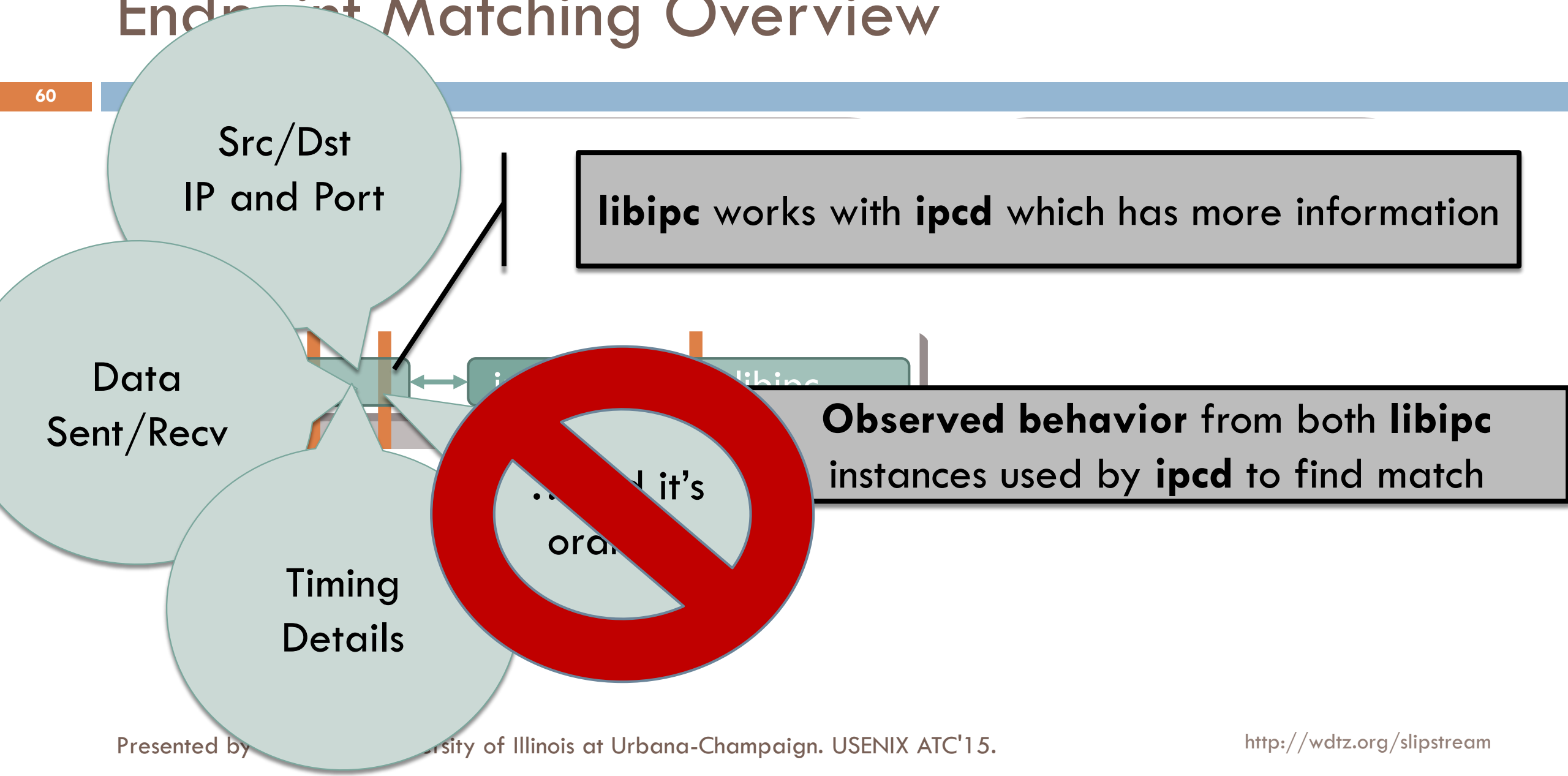


libipc works with **ipcd** which has more information

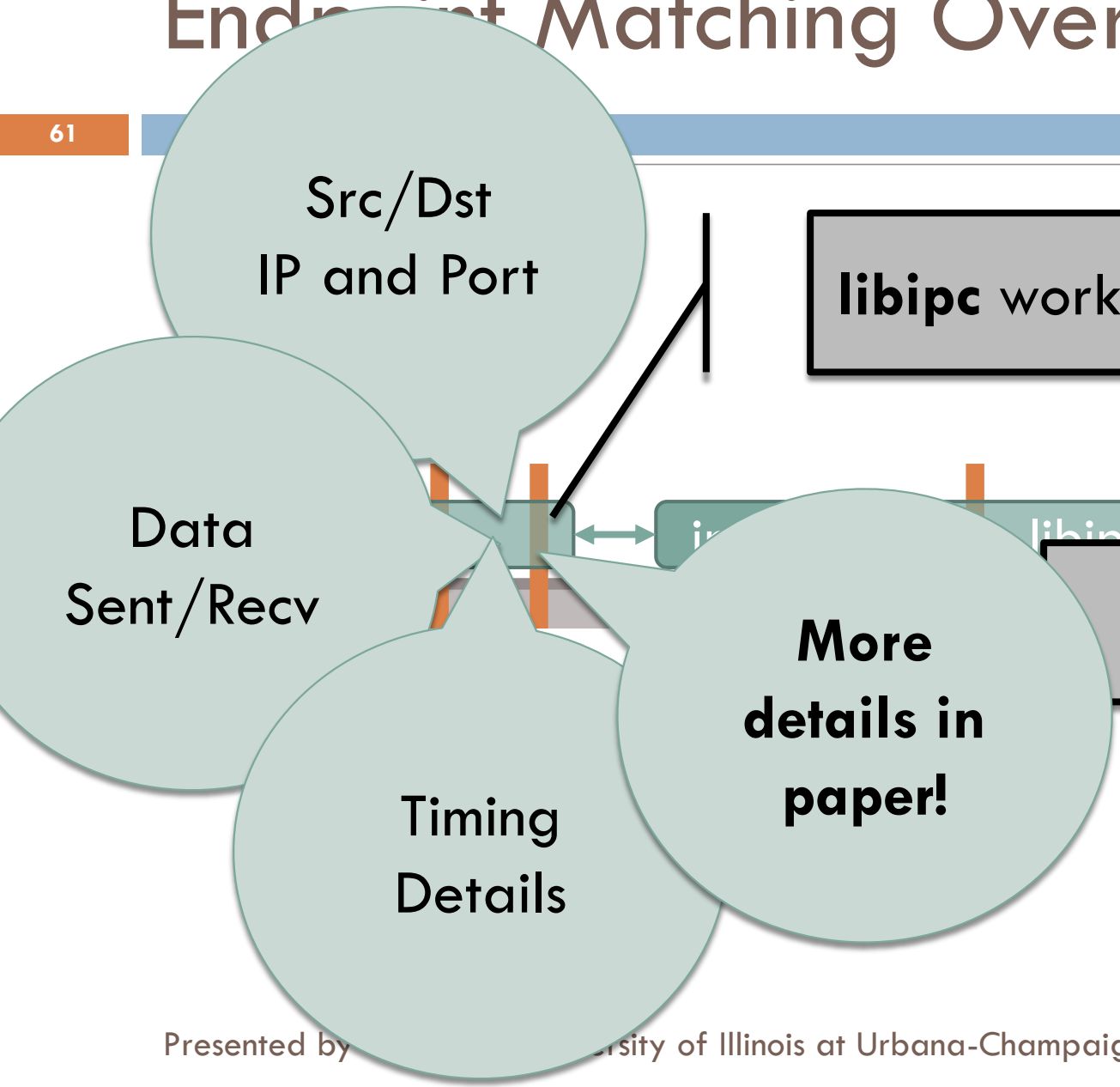
behavior from both **libipc**
ed by **ipcd** to find match

What information *can* be used to help pair endpoints?

Endpoint Matching Overview



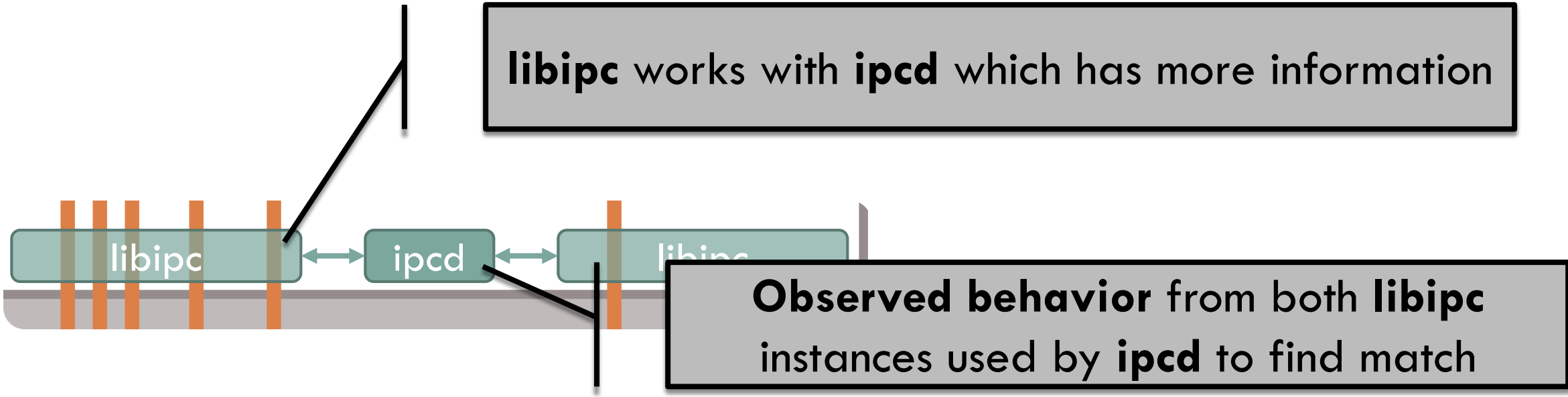
Endpoint Matching Overview



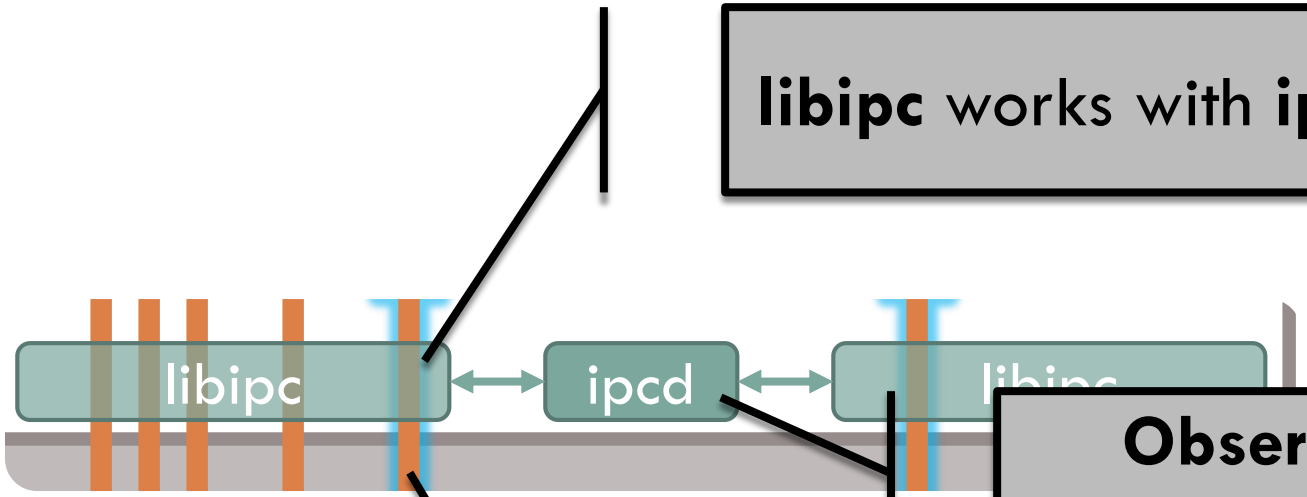
libipc works with **ipcd** which has more information

Observed behavior from both **libipc** instances used by **ipcd** to find match

Endpoint Matching Overview



Endpoint Matching Overview

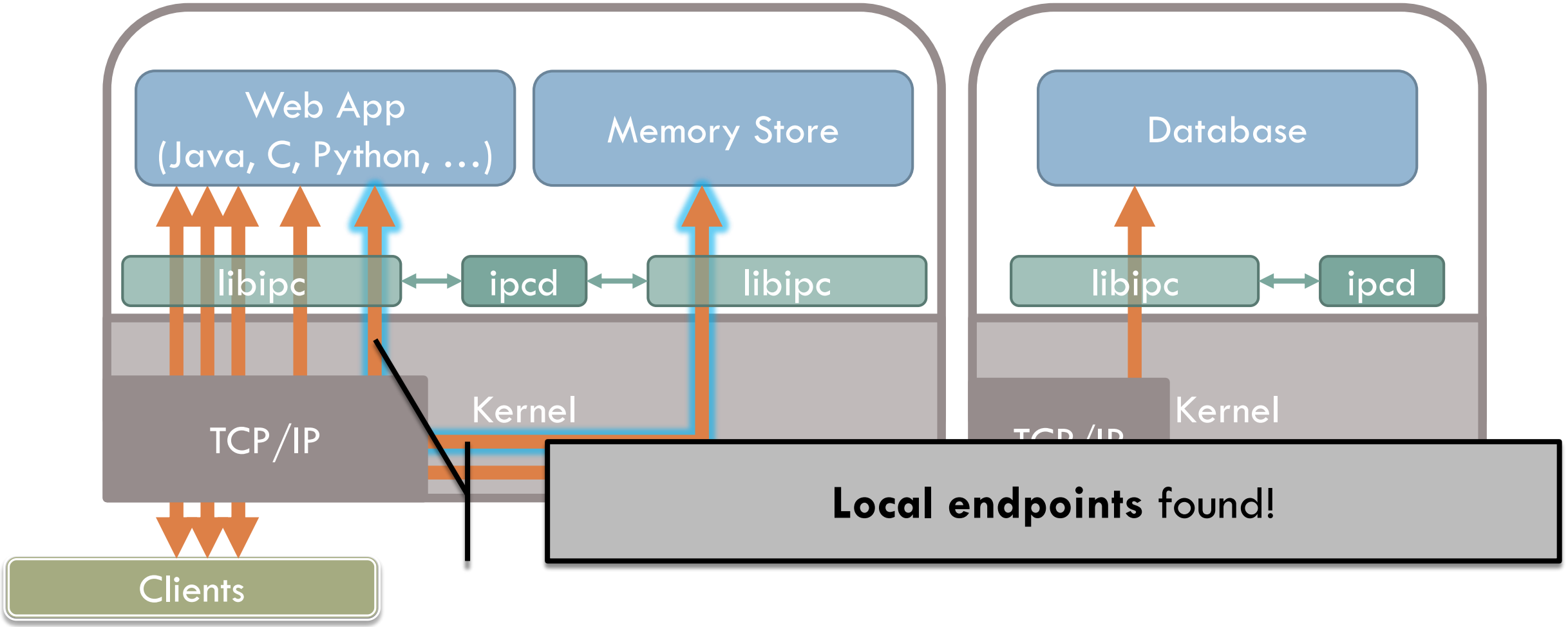


libipc works with **ipcd** which has more information

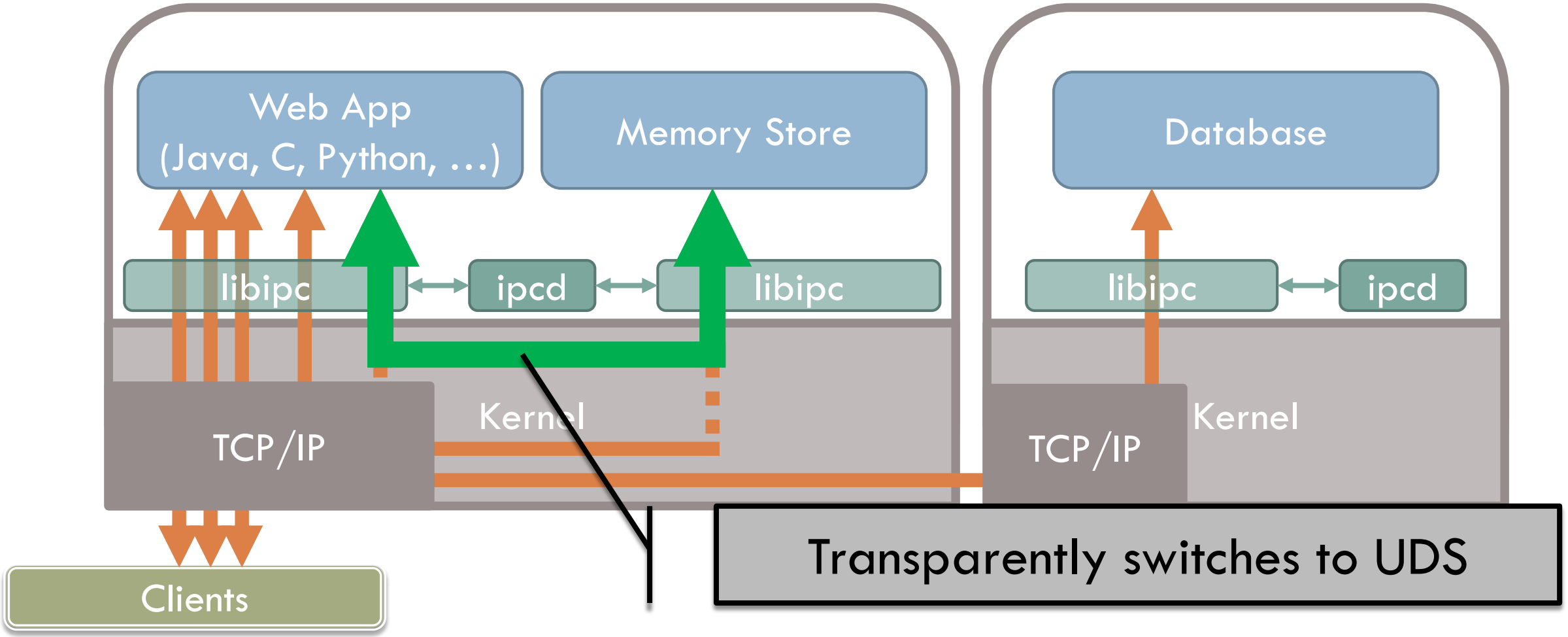
Observed behavior from both **libipc** instances used by **ipcd** to find match

Local endpoints found!

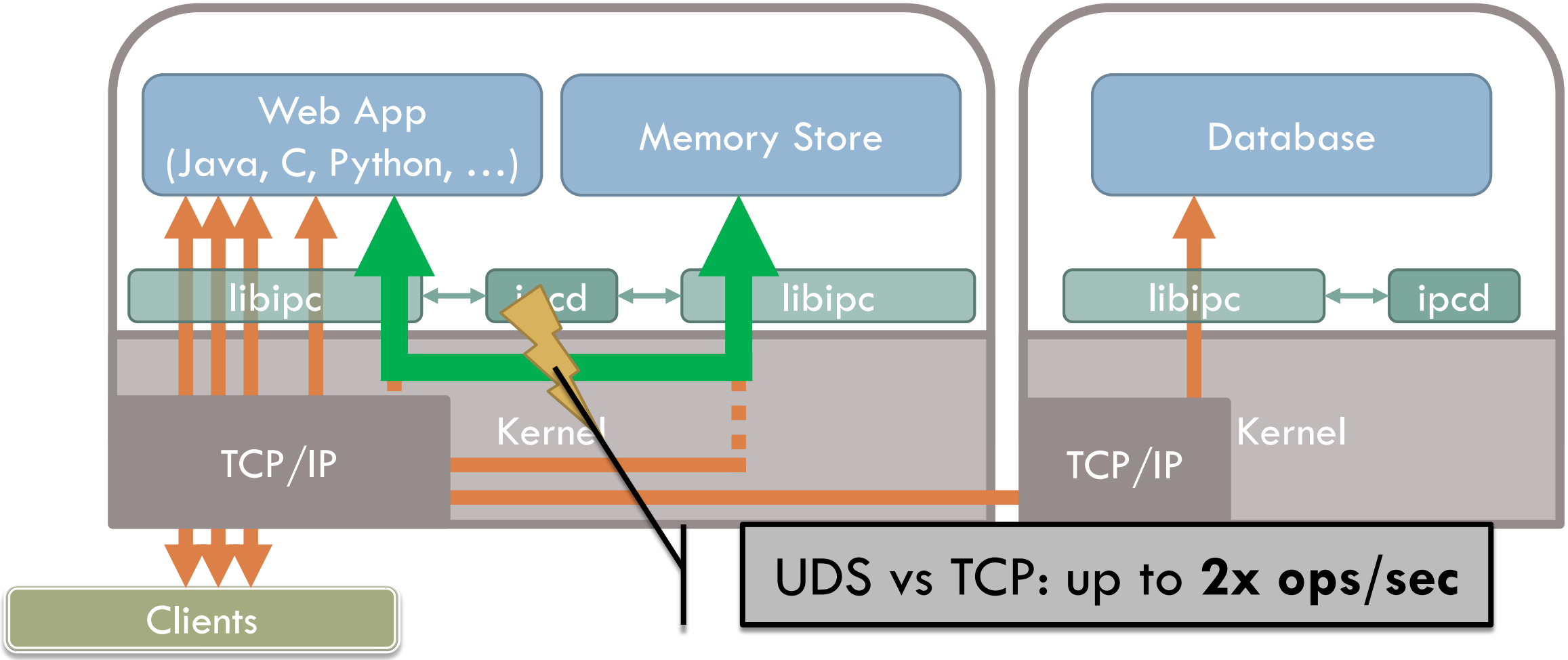
Endpoint Matching: Complete



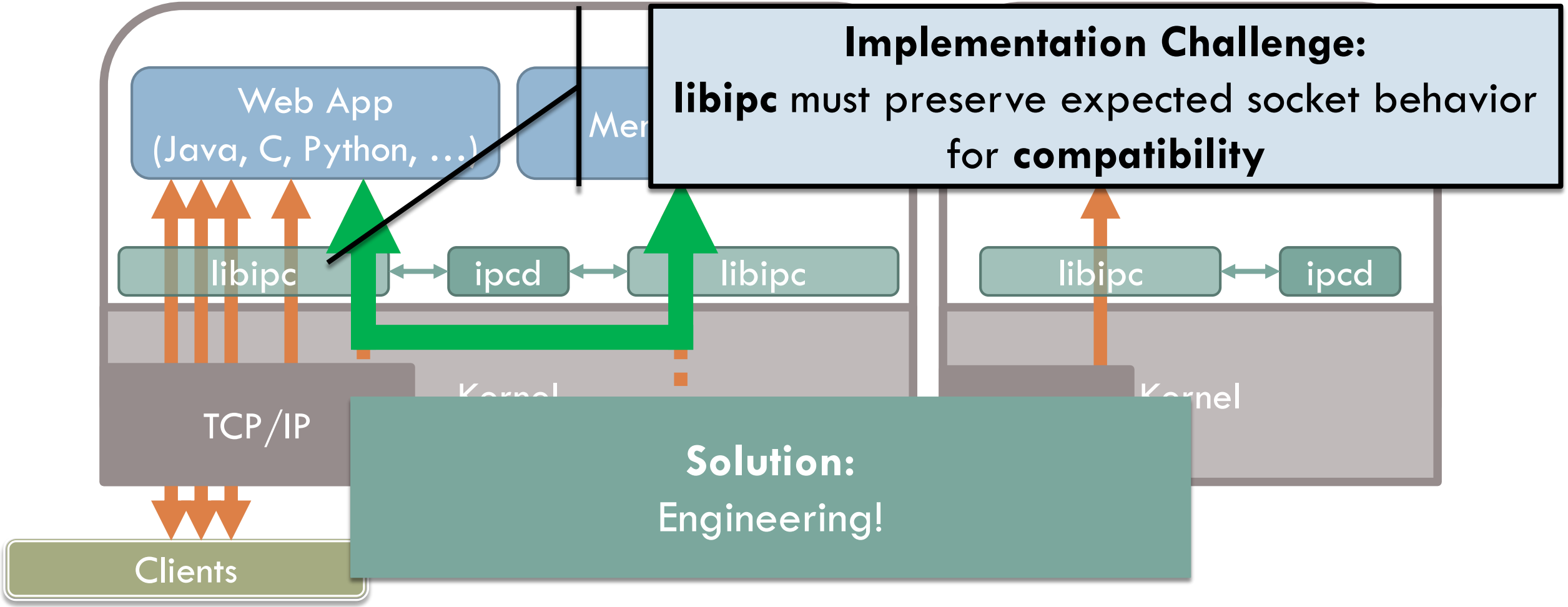
Automatic IPC Optimization



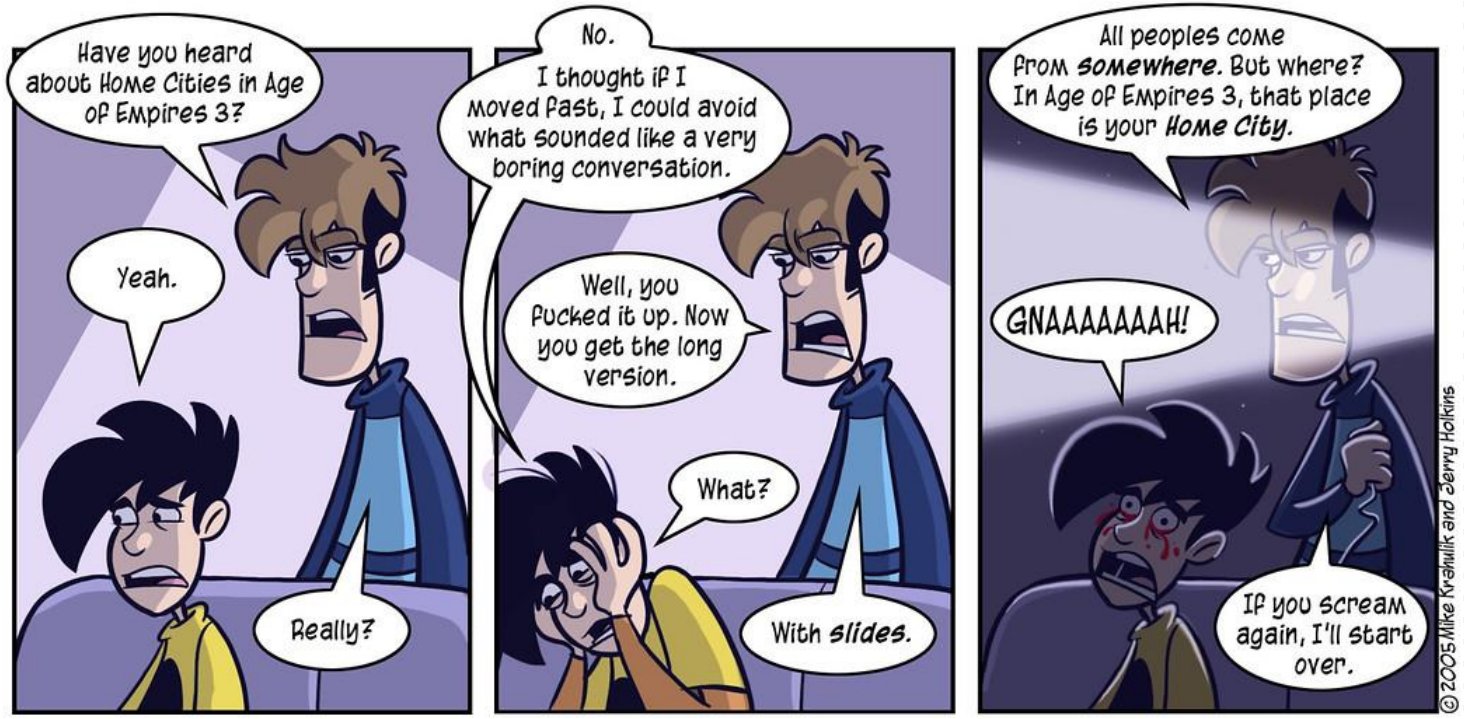
Automatic IPC Optimization



Without Modifying Applications?



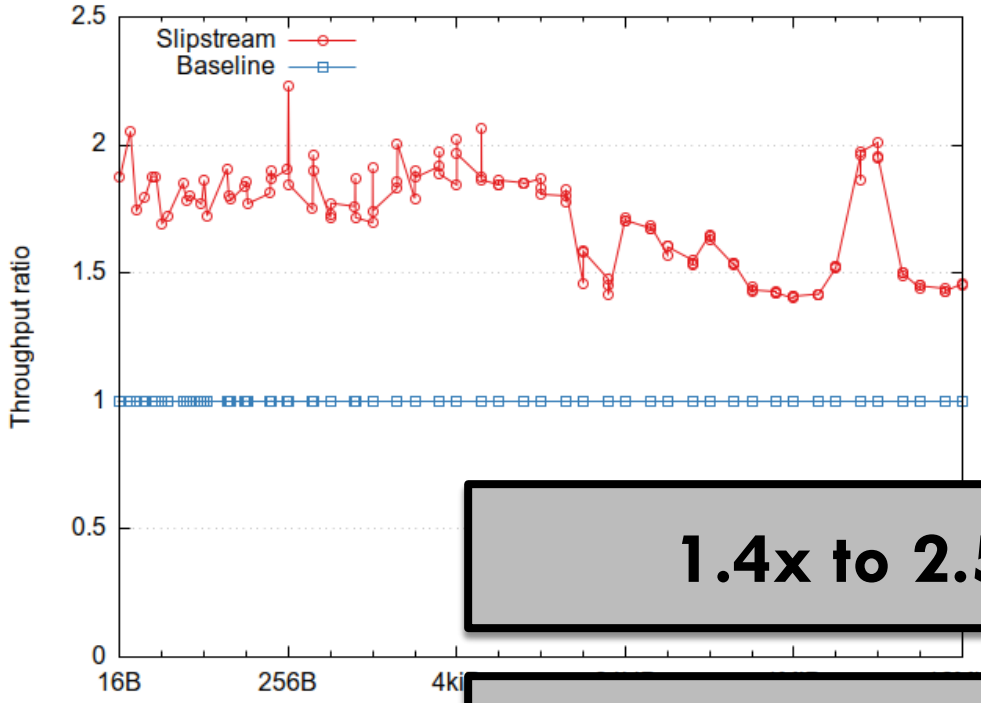
Results, with *slides*.



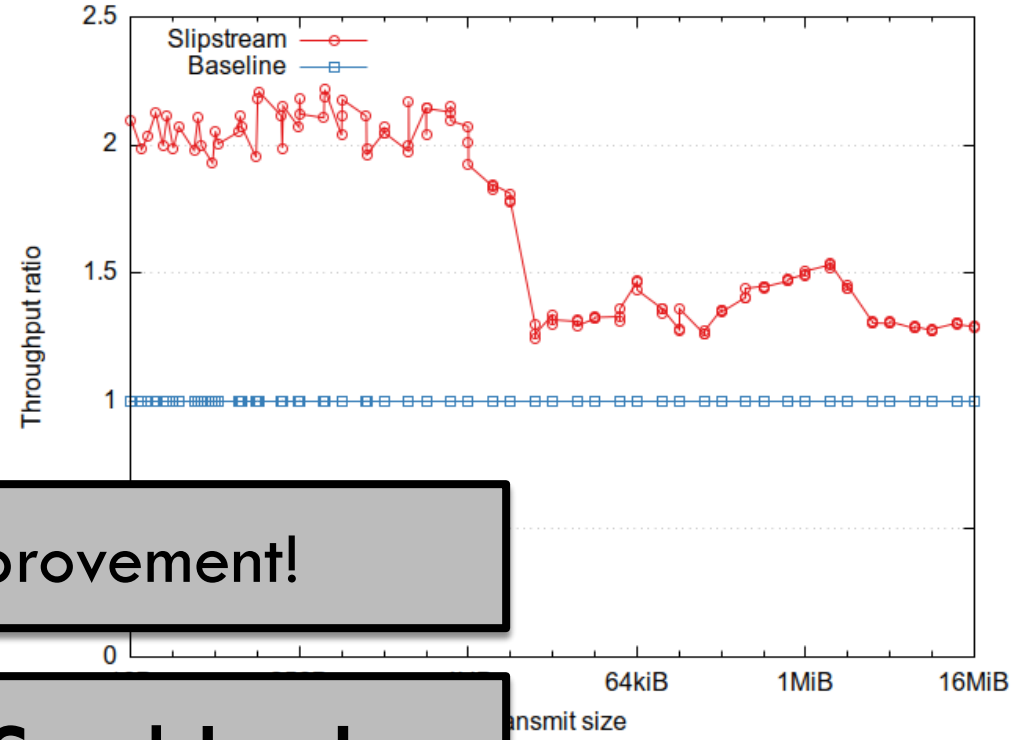
Source: http://www.penny-arcade.com/comic/2005/09/09

Faster than TCP? NetPIPE Microbenchmarks

NetPIPE-C



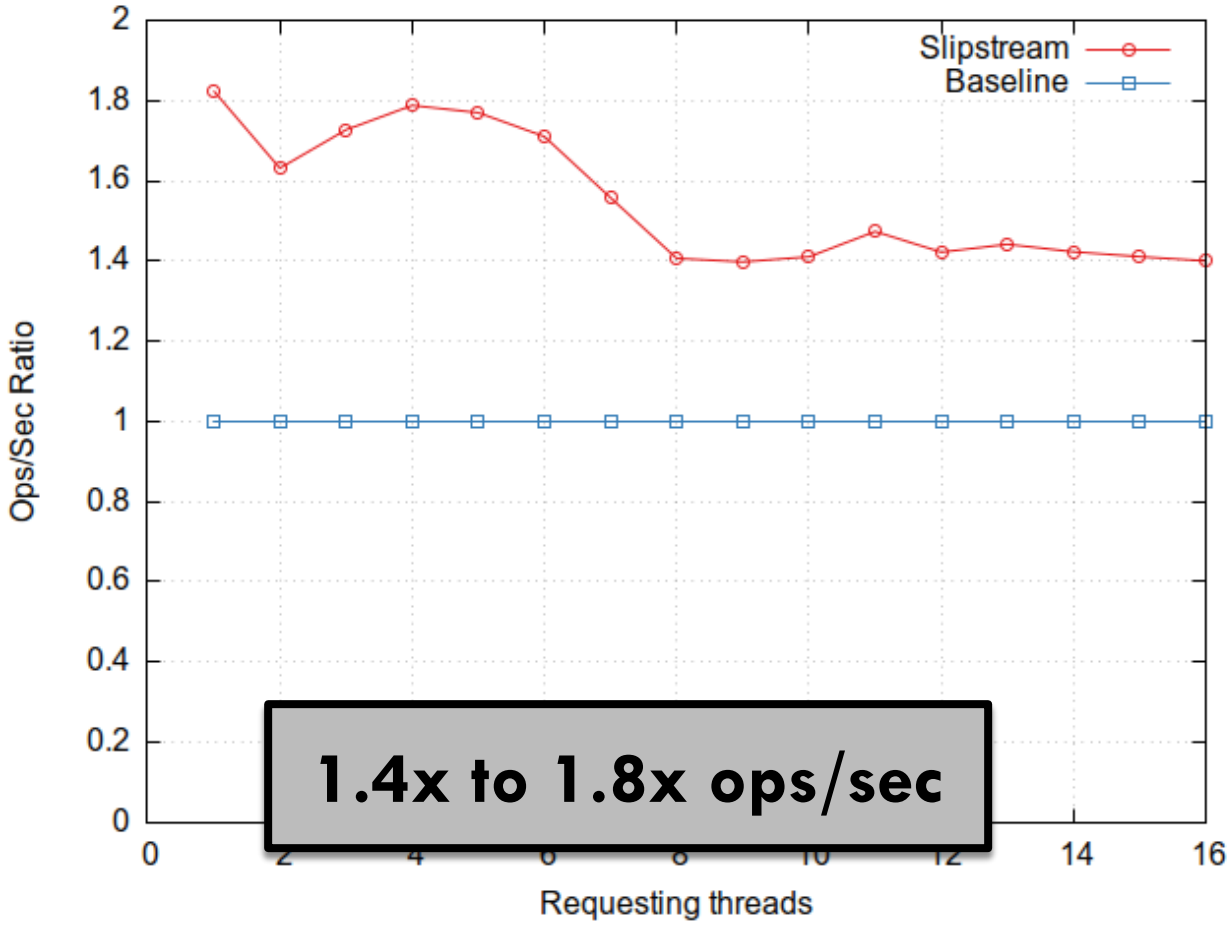
NetPIPE-Java



1.4x to 2.5x improvement!

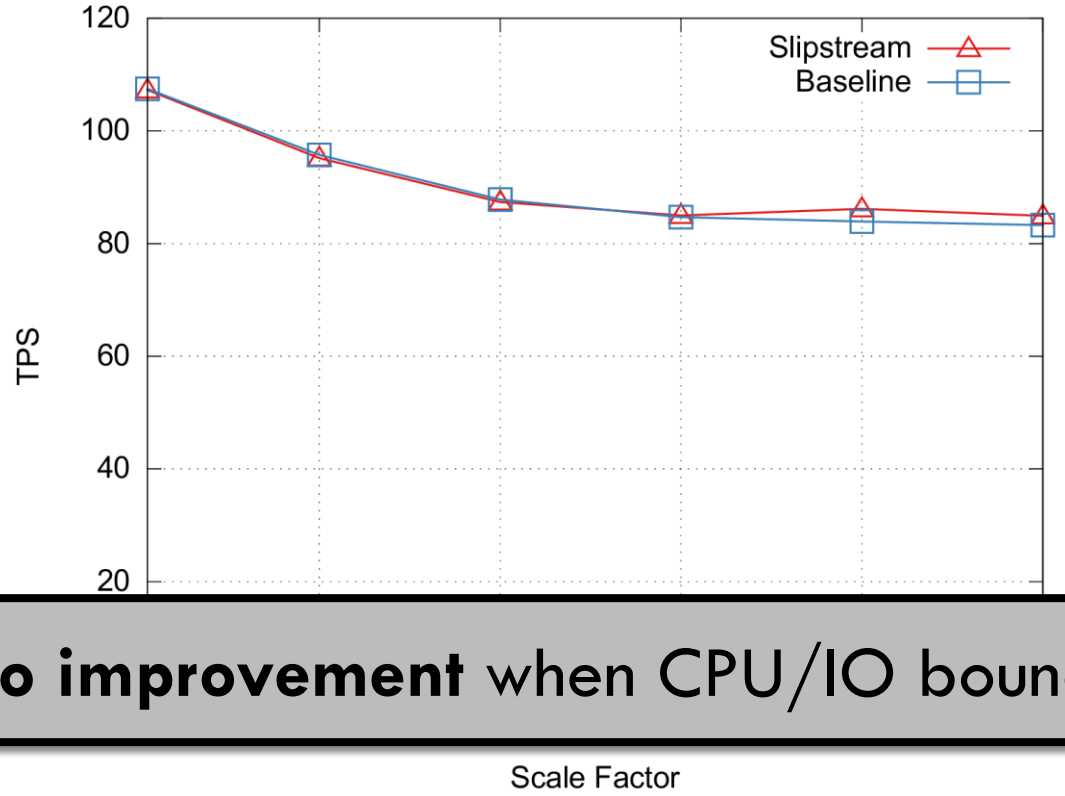
Works with both C and Java!

Faster than TCP? Memcached



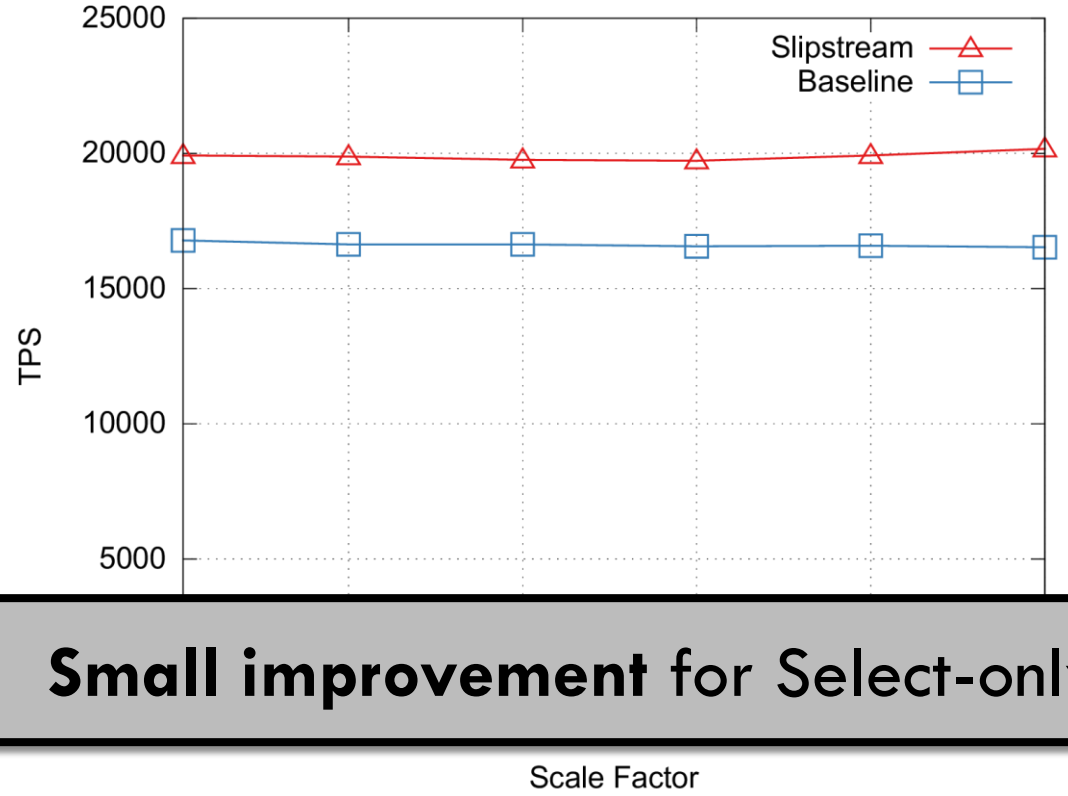
Faster than TCP? PostgreSQL

PostgreSQL – TPC-B



No improvement when CPU/IO bound

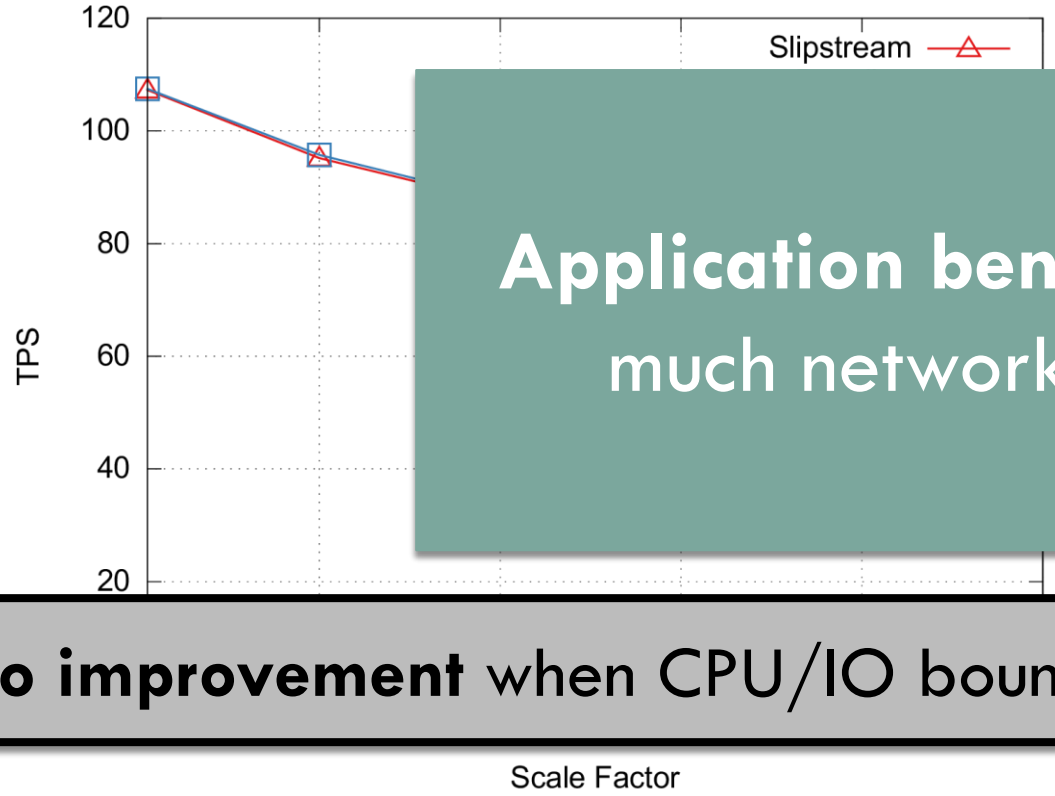
PostgreSQL - Select



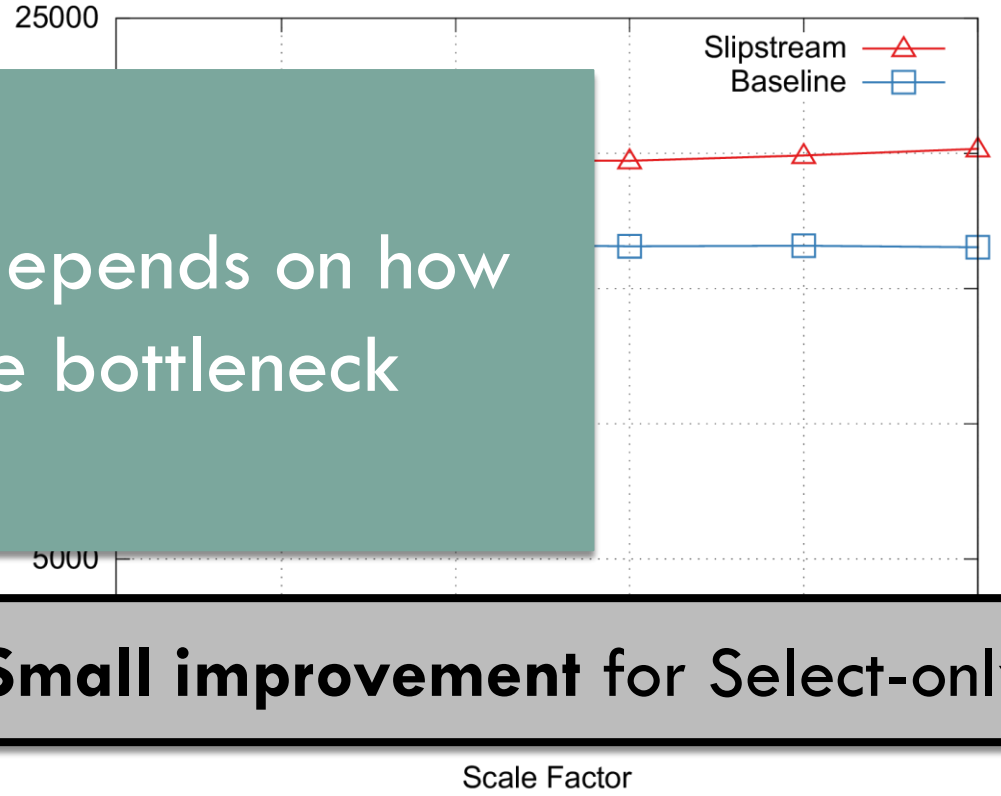
Small improvement for Select-only

Faster than TCP? PostgreSQL

PostgreSQL – TPC-B



PostgreSQL - Select



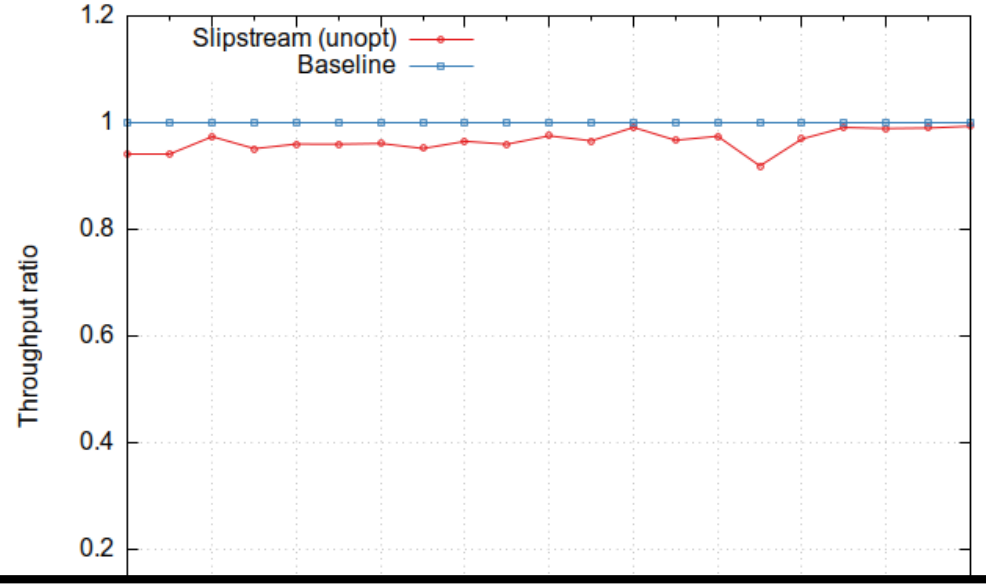
Application benefit depends on how much network is the bottleneck

No improvement when CPU/IO bound

Small improvement for Select-only

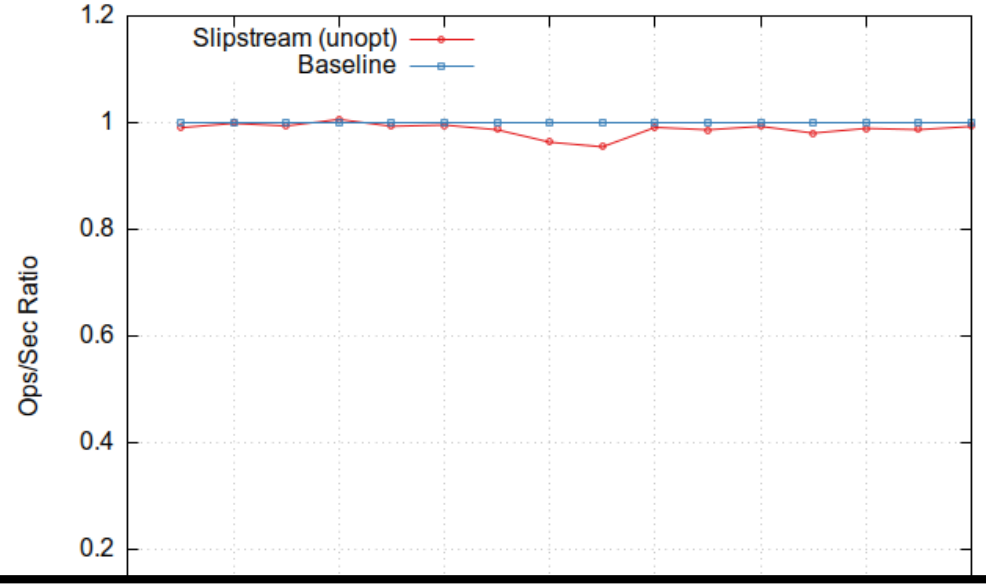
Slower than TCP? Slipstream Overhead

Netperf



~3.5% overhead for Netperf

Memcached

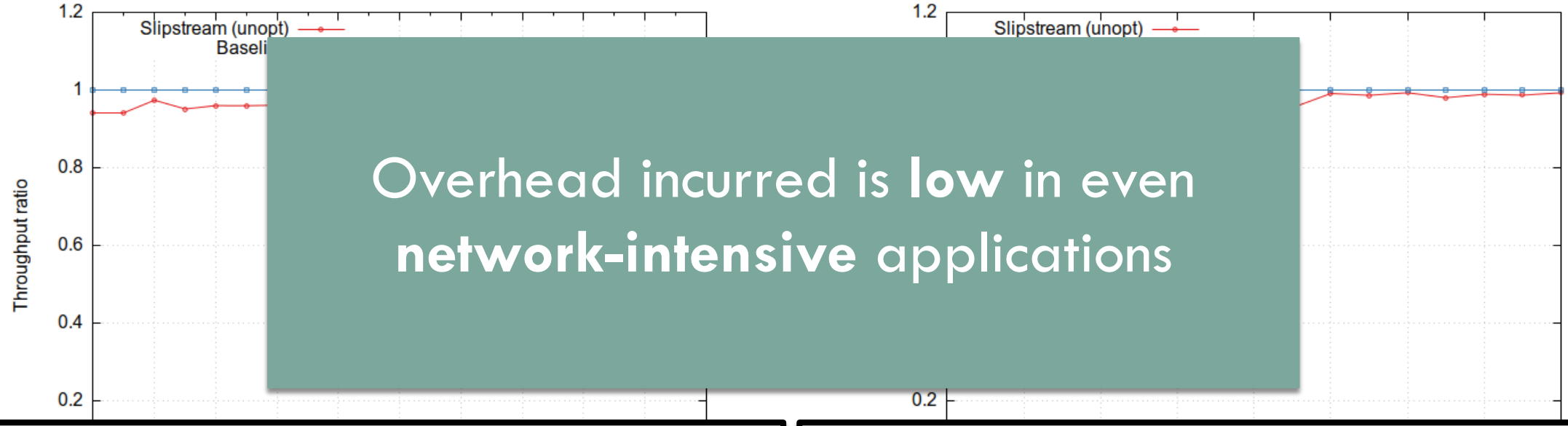


< 3% overhead for Memcached

Slower than TCP? Slipstream Overhead

Netperf

Memcached

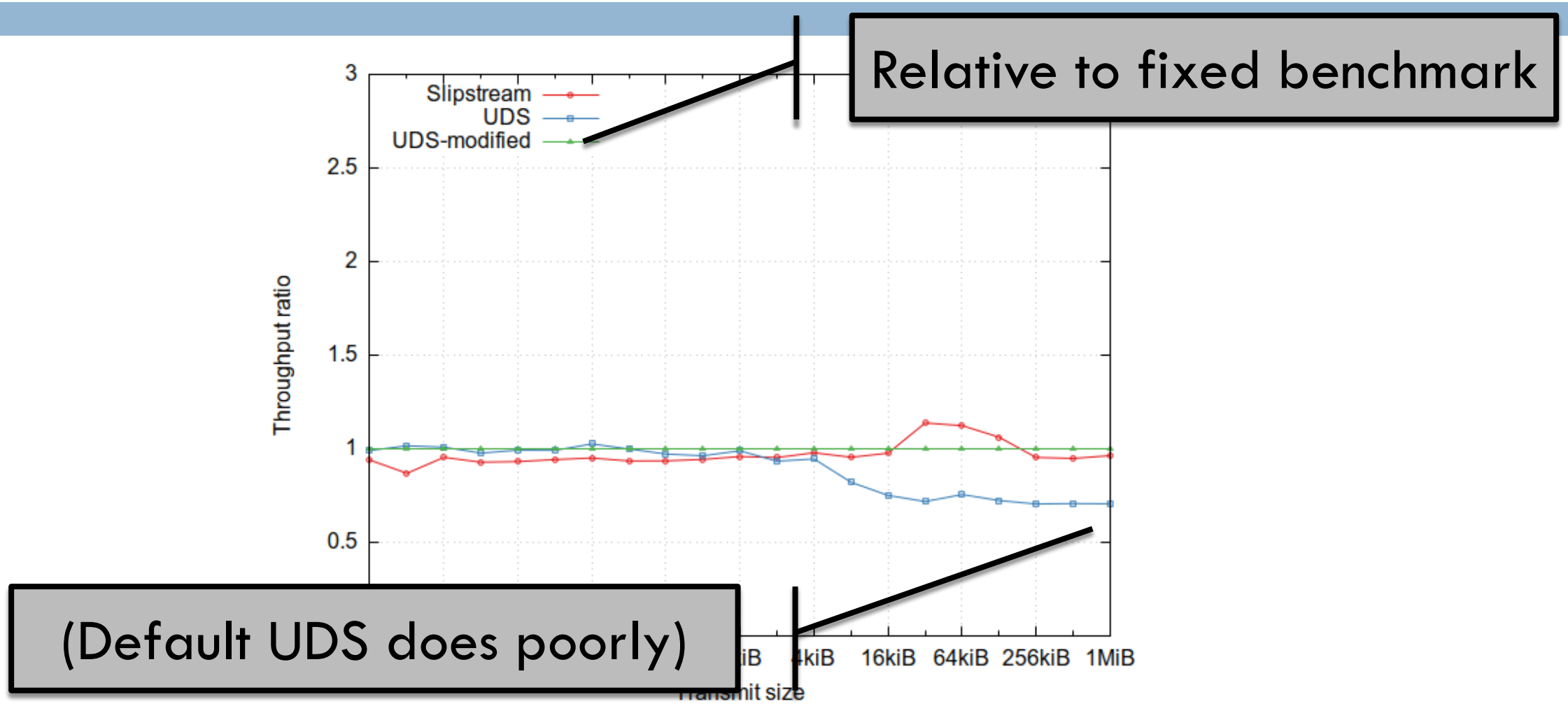


Overhead incurred is **low** in even **network-intensive** applications

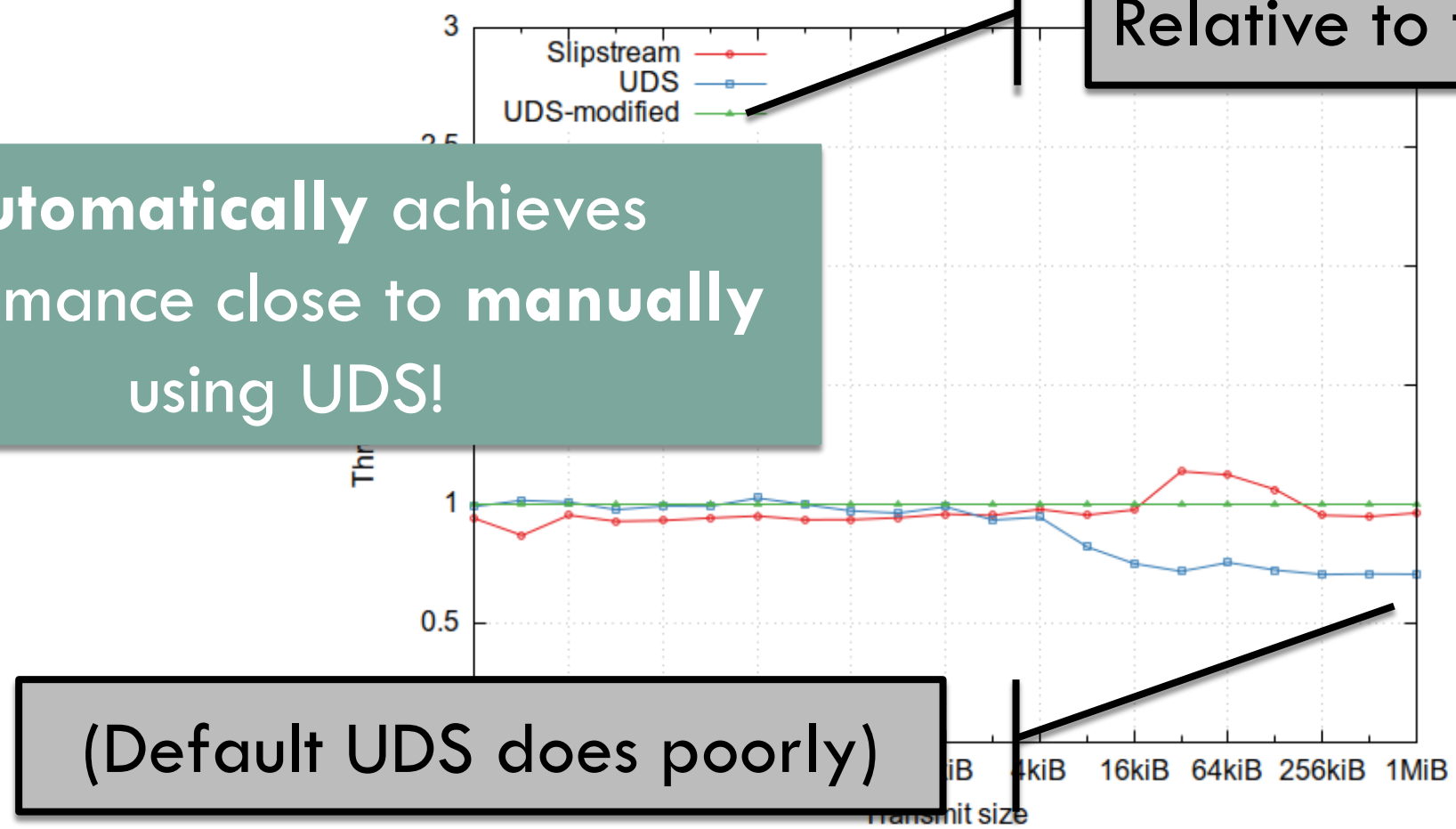
~3.5% overhead for Netperf

< 3% overhead for Memcached

Slower than manual UDS? Netperf Microbenchmark



Slower than manual UDS? Netperf Microbenchmark

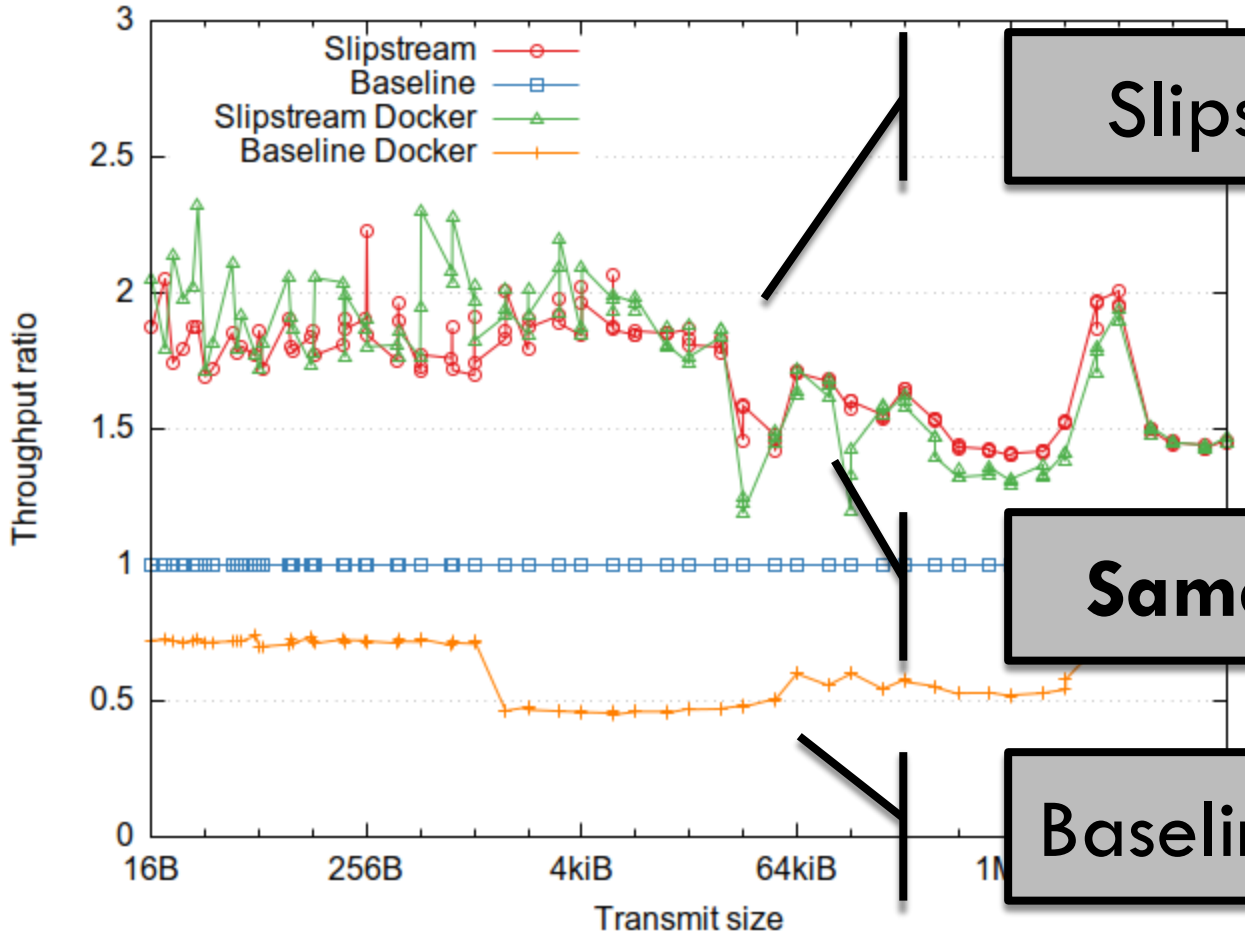


Relative to fixed benchmark

Automatically achieves performance close to **manually** using UDS!

(Default UDS does poorly)

Docker with NetPIPE-C



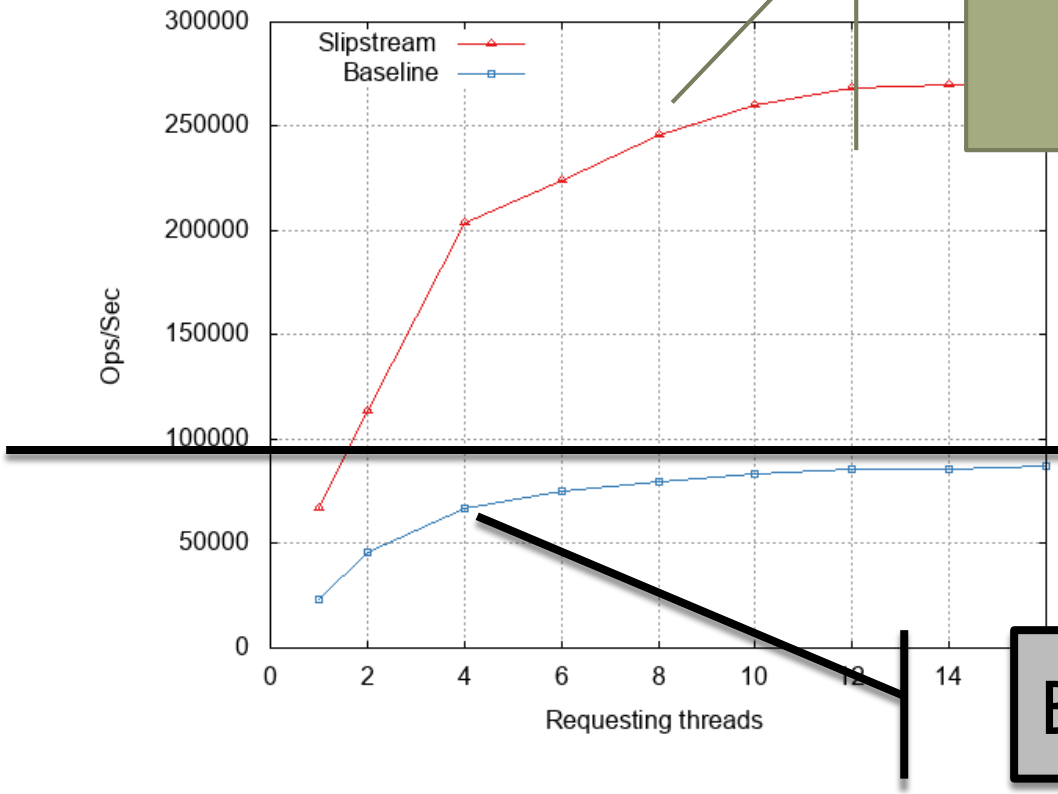
Slipstream shows **large speedups**

Same perf. with or without Docker!

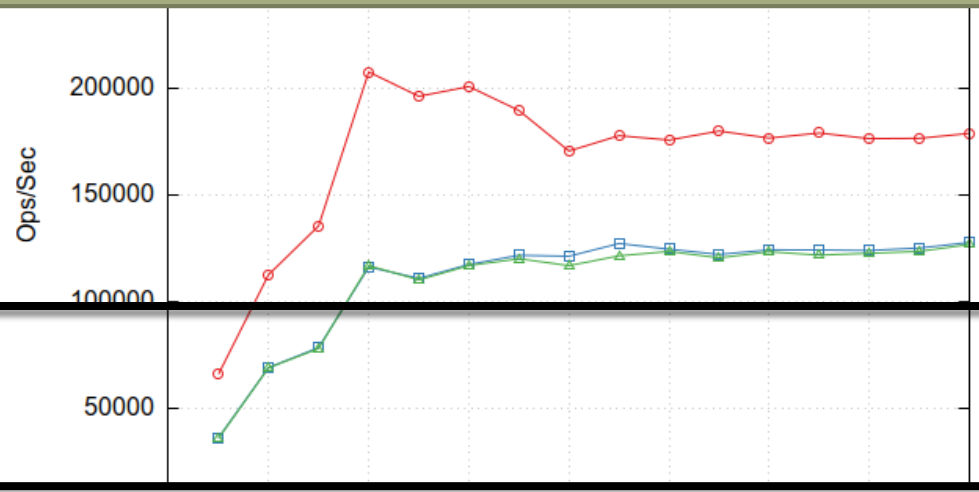
Baseline Docker is significantly slower

Docker with Memcached

Memcached with Docker



Curious:
Memcached with Slipstream **performs better** across Docker containers



Baseline Docker has lower performance

Software Compatibility

- **Supports many popular server applications**
  - MySQL, PostgreSQL, Redis, Memcached, Apache, Jenkins, …
- **Language agnostic** in practice
  - Java, C, Python, …
- And of course our microbenchmarks:
  - NetPIPE, netperf, iperf, lmbench

- **All work correctly with Slipstream**
- **All are optimized when communicating locally**

Slipstream Overview – Multiple Applications

