Distributed Timing Analysis in 100 Line Code

Tsung-Wei Huang

Research Assistant Professor

Department of Electrical and Computer Engineering

University of Illinois at Urbana-Champaign, IL, USA

What we Need Today

- **OpenTimer**
  - Static timing analysis (STA) tool
  - Installation guide (by Kunal Ghosh): https://www.udemy.com/vsd-a-complete-guide-to-install-open-source-eda-tools/
- **DtCraft**
  - Distributed programming system
  - Website: http://dtcraft.web.engr.illinois.edu/
  - GitHub: https://github.com/twhuang-uiuc/DtCraft
- **A Linux machine (Ubuntu recommended)**
  - G++ 7.2 (for C++17)
- **Demo code:** http://web.engr.illinois.edu/~thuang19/webinar.tar.gz

Install DtCraft

- **Download DtCraft**
  - http://dtcraft.web.engr.illinois.edu/download.html

```
~$ git clone https://github.com/twhuang-uiuc/DtCraft.git
~$ cd DtCraft
```

- **Build DtCraft**
  - Disable shared library for simplicity (--disable-shared)

```
~$ ./configure --disable-shared
~$ make
```

- **Make sure you have GCC/G++ 7 installed (C++17)**

```
~$ sudo apt-get update
~$ sudo apt-get install gcc-7 g++-7
```

Outline

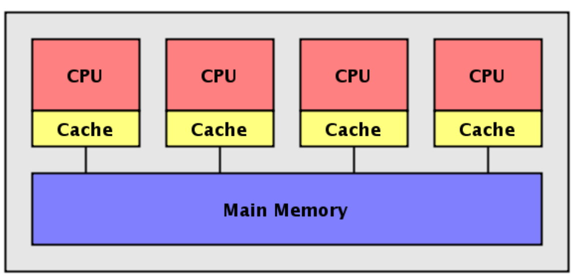
- **Express your parallelism in the right way**
  - A “hard-coded” distributed timing analysis framework
- **Boost your productivity in writing parallel code**
  - DtCraft system
- **Leverage your time to produce promising results**
  - Demo 1: A vanilla example
  - Demo 2: Distributed timing using DtCraft
  - Lab

Outline

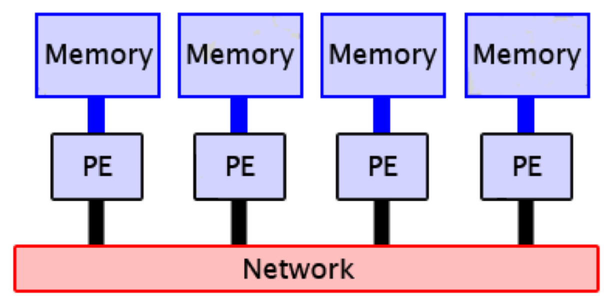
- **Express your parallelism in the right way**
  - A “hard-coded” distributed timing analysis framework
- **Boost your productivity in writing parallel code**
  - DtCraft system
- **Leverage your time to produce promising results**
  - Demo 1: A vanilla example
  - Demo 2: Distributed timing using DtCraft
  - Lab

Distributed Timing

- ❑ **Deal with the ever-increasing design complexity**
  - ❑ Billions of transistors result in very large timing graphs
  - ❑ Analyze the timing under different conditions
  - ❑ Vertical scaling is not cost efficient
- ❑ **Want to scale out our computations**
  - ❑ Leverage the power of computer clusters (cloud computing)



Single node (threaded)

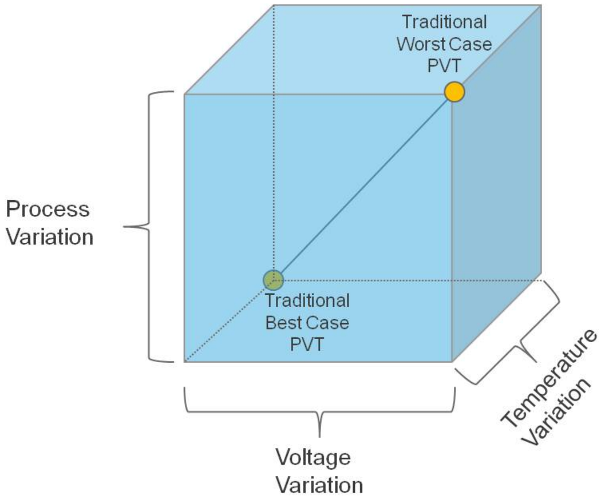


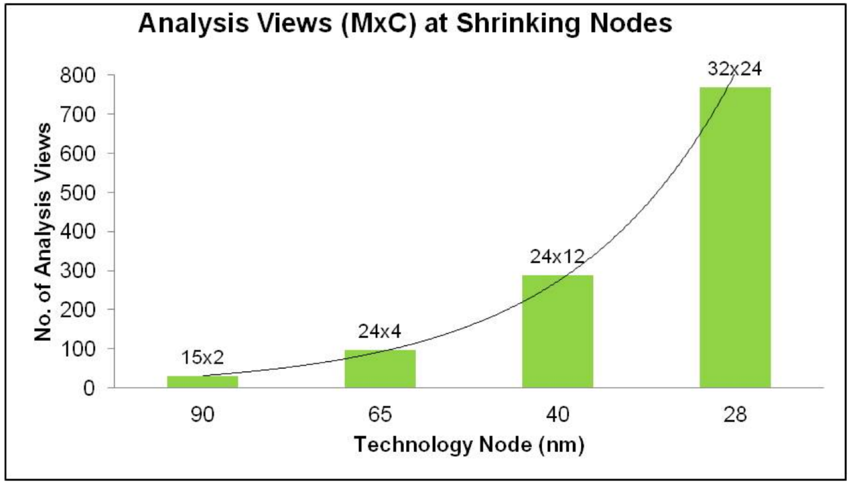
Datacenter (distributed)

Motivation: Speed up Timing Closure

- ❑ **Multi-mode multi-corner (MMMC) timing analysis**
  - ❑ Test modes, functional modes
  - ❑ Process, voltage, temperature (PVT)
  - ❑ Timing runs across all combinations
    - Temperature: Tmax/Tmin
    - Volgate: Vmin/Vmax

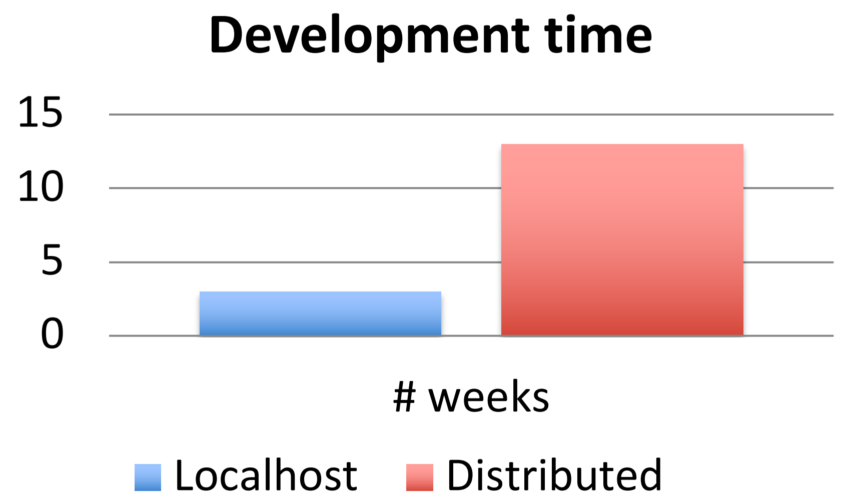
    2x2 combinations (timing views/reports)
  - ❑ Each combination is referred to as a *timing view*





Good News and Bad News

- **Each timing view is logically parallel to each other**
- **Developing a distributed program is very difficult**
  - Several weeks more than a single-machine counterpart
  - Network programming, subtly buggy code, etc
- **Scalability and transparency**
  - Intend to focus on high level rather than low level
  - Want better productivity
  - Want better flexibility
  - Want better performance



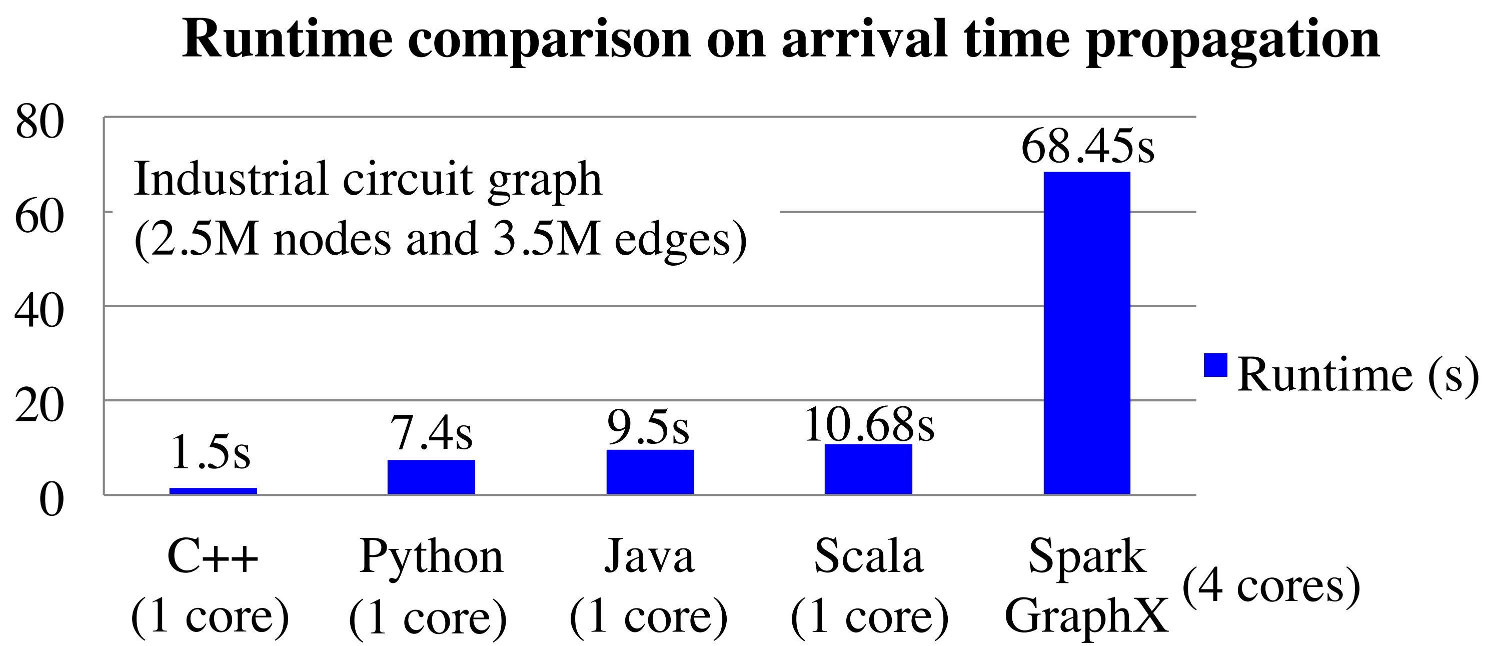
Distributed Systems in Big Data

- **Hadoop**
  - Distributed MapReduce platform on HDFS
- **Cassandra**
  - Scalable multi-master database
- **Chukwa**
  - A distributed data collection system
- **Zookeeper**
  - Coordination service for distributed application
- **Mesos**
  - A high-performance cluster manager
- **Spark**
  - A fast and general computing engine for big-data analytics

The Questions are

- ❑ Are these packages suitable for timing?
- ❑ What are the potential hurdles for EDA to use them?
- ❑ How much code rewrite do I need?
- ❑ What is the significance of adopting new languages?
- ❑ Will I lose performance?

Big-data Tool is Not an Easy Fit!



Method	Spark (RDD + GraphX Pregel)	Java (SP)	C++ (SP)
Runtime (s)	68.45	9.5	**1.50**

Overhead of big data tools

Language difference

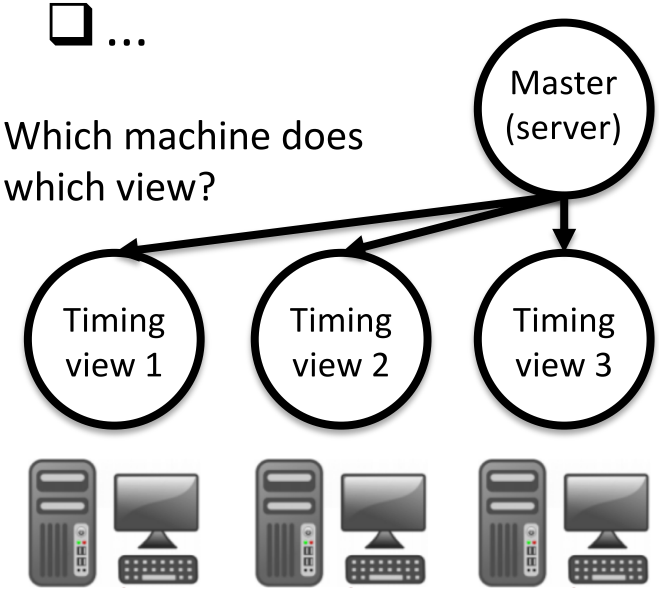
A Hard-coded Distributed MMMC Framework

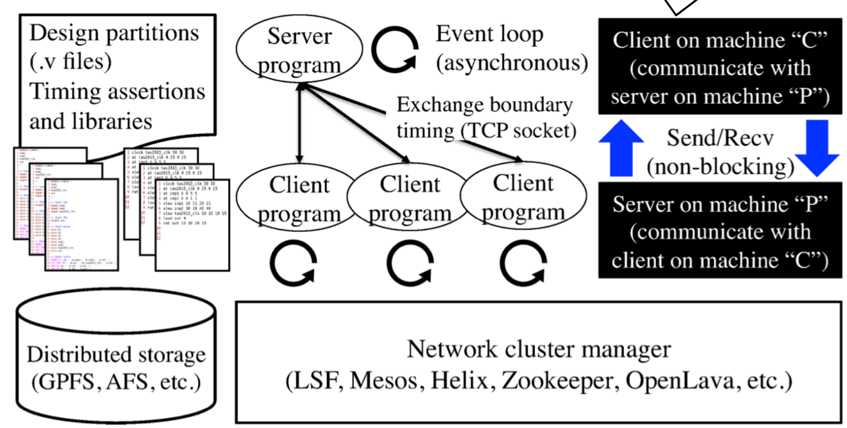
- Built from the scratch using raw Linux Socket
  - Hard code using Linux sockets
  - Explicit data movement
  - Explicit job execution
  - Explicit parallelism management
  - …

Difficult scalability ☹
Large amount of code rewrites ☹

- Non-blocking IO
- Event-driven programming
- Serialization/Deserialization

Which machine does which view?





Huang et al., "A Distributed Timing Analysis Framework for Large Designs," IEEE/ACM DAC16

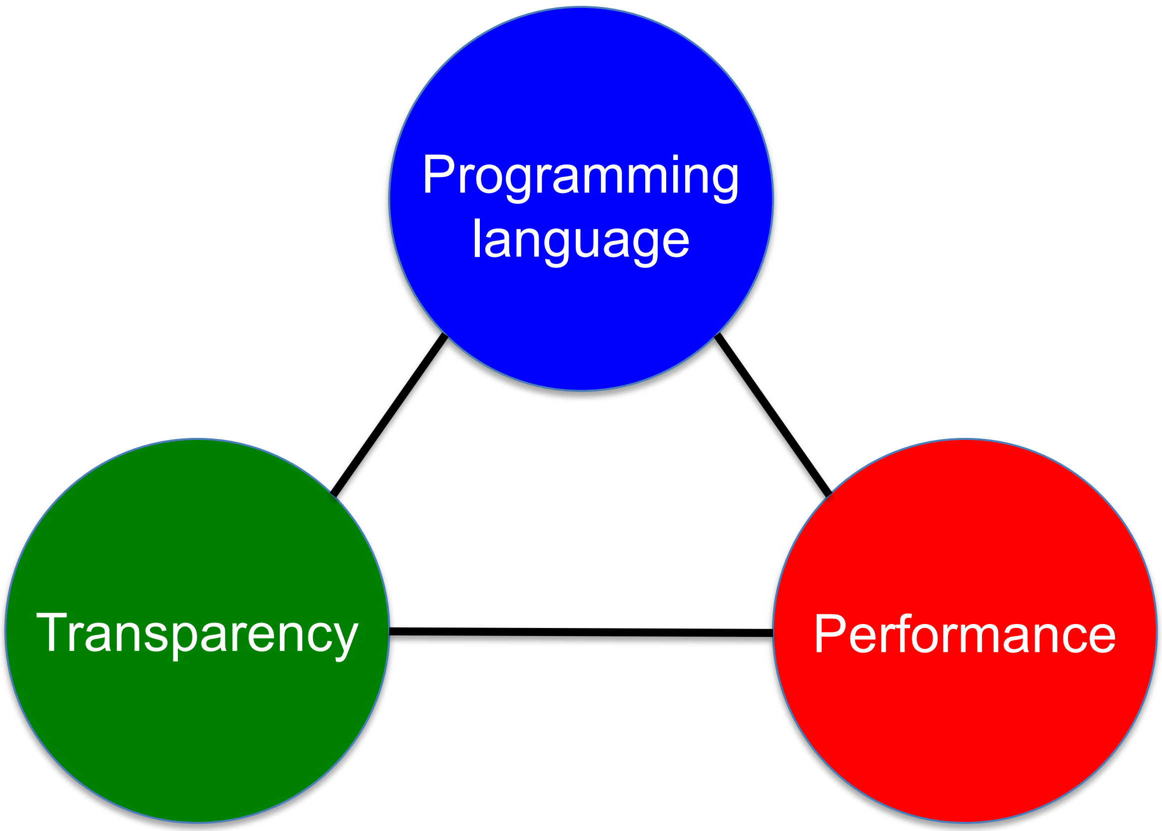
Observations

- **Big data doesn't fit well in timing**
  - IO-bound vs CPU-bound
  - Unstructured data vs structured data
  - JVM vs C/C++
- **Life shouldn't be hard-coded**
  - Deal with low-level socket programming
  - Move data explicitly between compute nodes
  - Manage cluster resources on your own
  - Result in a large amount of development efforts
- **Want parallel programming *at scale* more *productive***

Outline

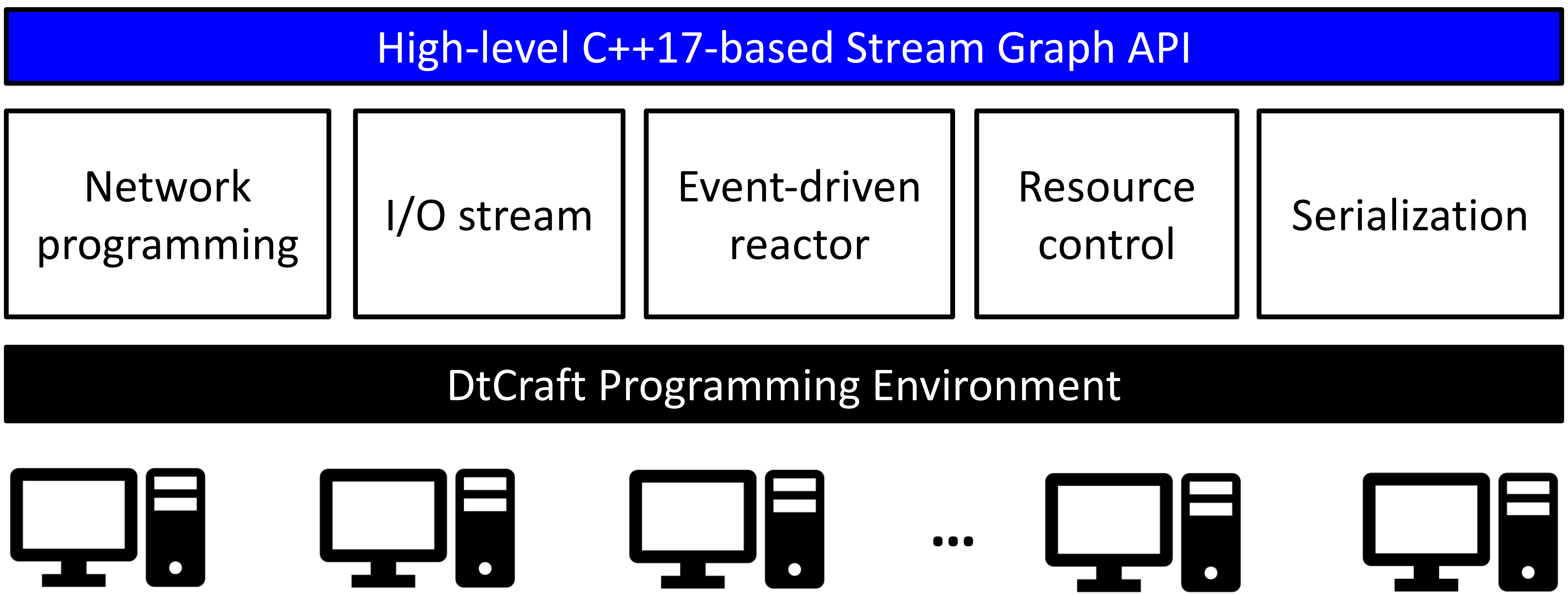
- ❑ **Express your parallelism in the right way**
  - ❑ A "hard-coded" distributed timing analysis framework
- ❑ **Boost your productivity in writing parallel code**
  - ❑ DtCraft system
- ❑ **Leverage your time to produce promising results**
  - ❑ Demo 1: A vanilla example
  - ❑ Demo 2: Distributed timing using DtCraft
  - ❑ Lab

What does “Productivity” really mean?



A New Solution: DtCraft

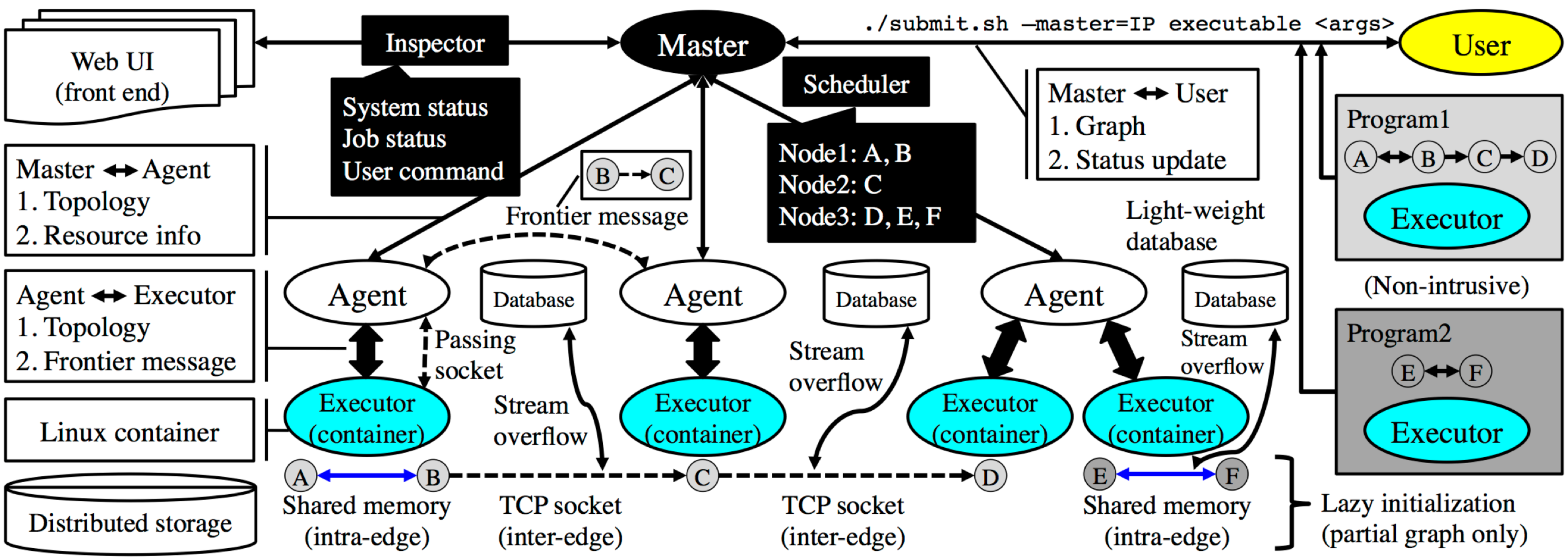
- ❑ **A unified engine to simplify cluster programming**
  - ❑ Completely built from the ground up using C++17
- ❑ **Save your time away from the pain of DevOps**



T.-W. Huang, C.-X. Lin, and M. D. F. Wong, "DtCraft: A high-performance distributed execution engine at scale," IEEE TCAD, to appear, 2018

System Architecture

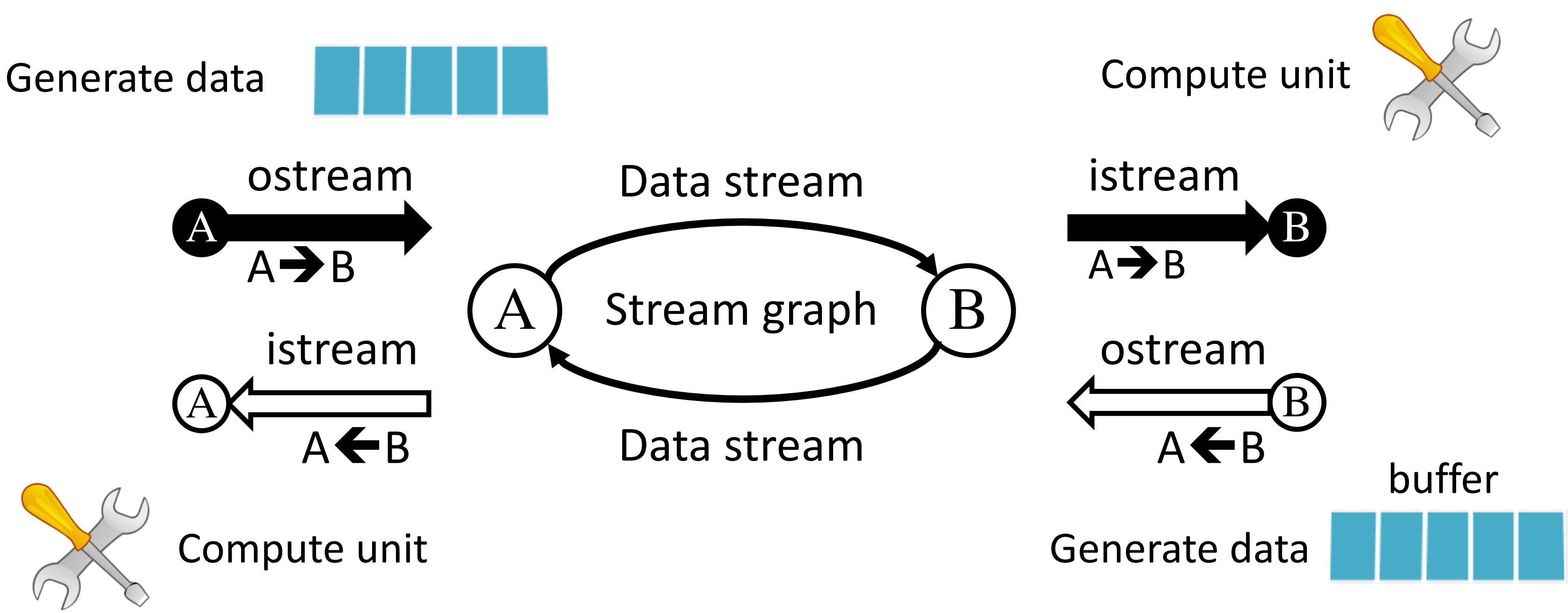
- ❑ **Express your parallelism in our *stream graph* model**
  - ❑ Generic dataflow at any granularity
- ❑ **Deliver transparent concurrency through the kernel**
  - ❑ Automatic workload distribution and message passing



DtCraft website: http://dtcraft.web.engr.illinois.edu/
DtCraft github: https://github.com/twhuang-uiuc/DtCraft

Stream Graph Programming Model

- ❑ **A general representation of a dataflow**
  - ❑ Abstraction over computation and communication
- ❑ **Analogous to the assembly line model**
  - ❑ Vertex storage → goods store
  - ❑ Stream processing unit → independent workers



Outline

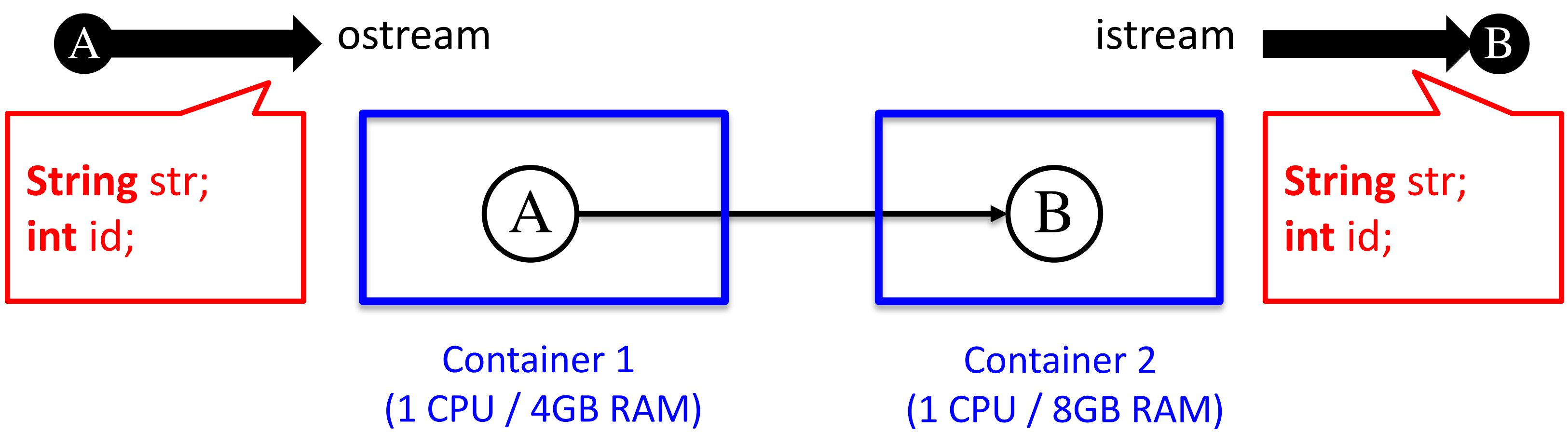
- ❑ **Express your parallelism in the right way**
  - ❑ A “hard-coded” distributed timing analysis framework
- ❑ **Boost your productivity in writing parallel code**
  - ❑ DtCraft system
- ❑ **Leverage your time to produce promising results**
  - ❑ Demo 1: A vanilla example
  - ❑ Demo 2: Distributed timing using DtCraft
  - ❑ Lab

Write a DtCraft Application

- ❑ **Step 1: Decide the stream graph of your application**
- ❑ **Step 2: Specify the data to stream between vertices**
- ❑ **Step 3: Define the stream computation callback**
- ❑ **Step 4: Attach resources on vertices (optional)**
- ❑ **Step 5: Submit**

./submit –master=host hello-world

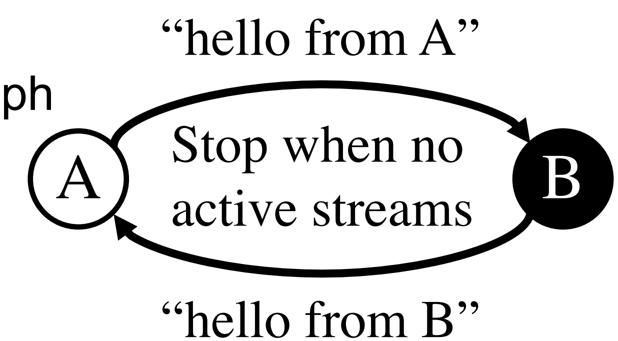
```
auto L = [=] (auto& vertex, auto& istream) {
  if(string data; istream(data) != -1) {
    // Your program control flow.
  } else { ... }
};
```



A Vanilla Example

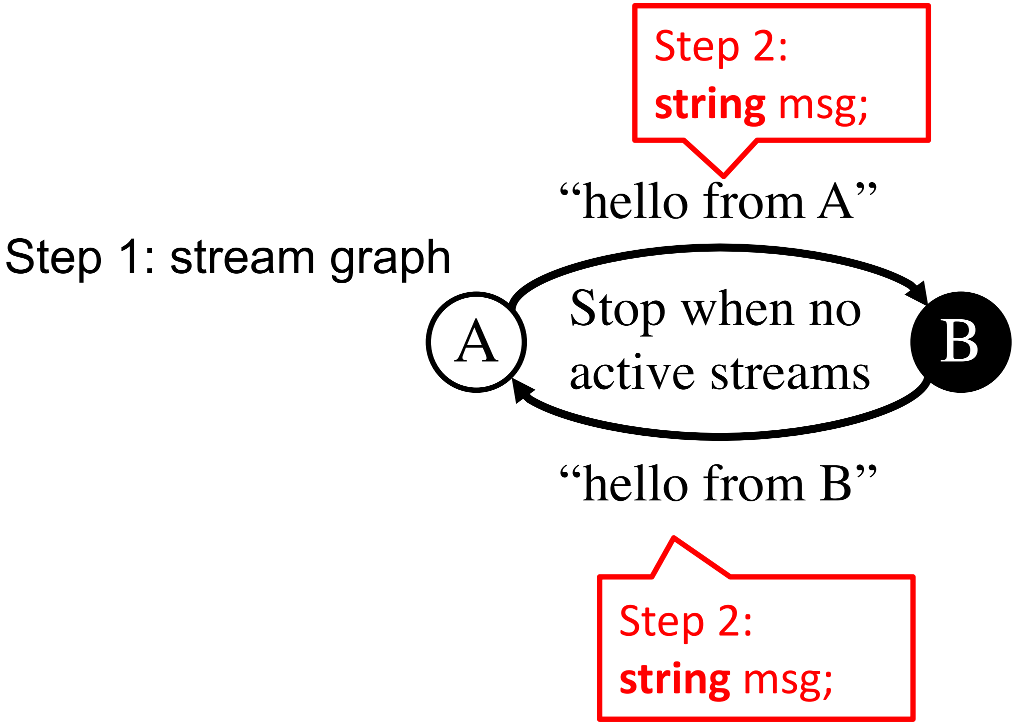
- **A cycle of two vertices and two streams**
  - Each vertex sends a hello message to the other
  - Closes the underlying stream channel

Step 1: stream graph



A Vanilla Example

- A cycle of two vertices and two streams
  - Each vertex sends a hello message to the other
  - Closes the underlying stream channel



A Vanilla Example

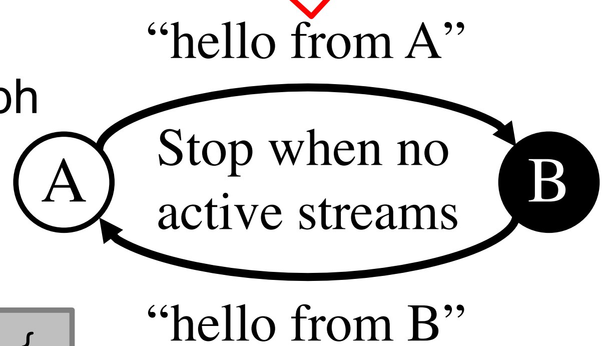
- ❑ **A cycle of two vertices and two streams**
  - ❑ Each vertex sends a hello message to the other
  - ❑ Closes the underlying stream channel

Step 2:
string msg;

Step 3: A➔B callback

```
[=] (auto& B, auto& is) {
  Extract string from is;
  print string;
}
```

Step 1: stream graph

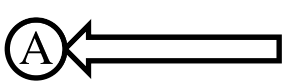


istream 

Step 3: A🡐B callback

```
[=] (auto& A, auto& is) {
  Extract string from is;
  print string
}
```

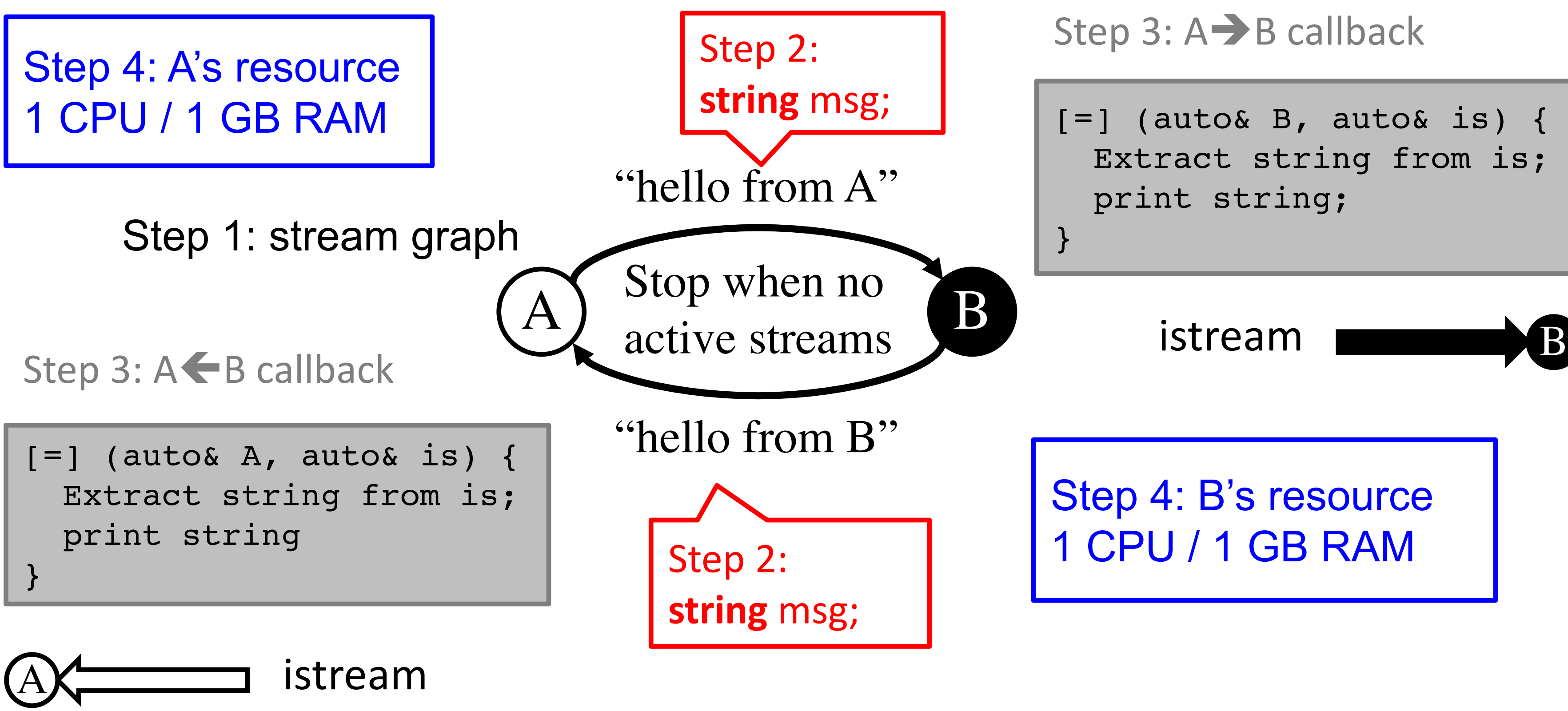
Step 2:
string msg;



istream

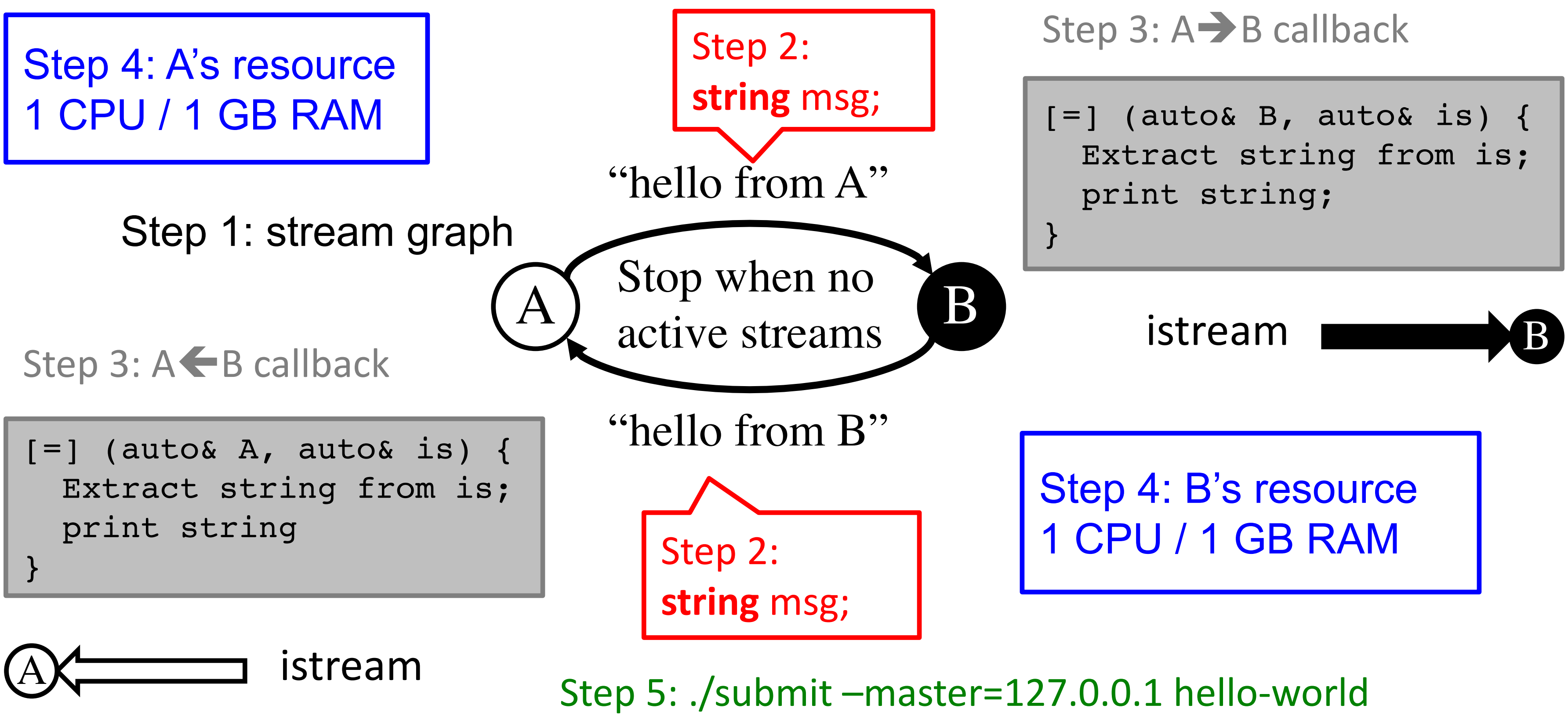
A Vanilla Example

- A cycle of two vertices and two streams
  - Each vertex sends a hello message to the other
  - Closes the underlying stream channel



A Vanilla Example

- ❑ **A cycle of two vertices and two streams**
  - ❑ Each vertex sends a hello message to the other
  - ❑ Closes the underlying stream channel



Demo (hello_world.cpp)

- **Hello world**
  - Create a stream graph of two vertices and two streams
  - Use container interface to manage cluster resources
- **Local mode execution**
  - Single process
- **Distributed mode execution**
  - Launch master and agents to set up a DtCraft cluster
  - Submit hello_world to the cluster

Login user	host	DtCraft home
twhuang	csl-408-08.csl.Illinois.edu	/home/twhuang/DtCraft
twhuang	csl-408-14.csl.Illinois.edu	/home/twhuang/DtCraft

Notice: Replace with your own login/hosts/DtCraftHome.

Demo code: http://web.engr.illinois.edu/~thuang19/webinar.tar.gz

Debrief

```
dtc::Graph G;

auto A = G.vertex();
auto B = G.vertex();
```

Vertex

```
G.container().add(A).cpu(1).memory(1_GB);

auto AB = G.stream(A, B).on(
  [] (dtc::Vertex& B, dtc::InputStream& is) {
    if(std::string b; is(b) != -1) {
      dtc::cout("Received: ", b, '\n');
      return dtc::Event::REMOVE;
    }
    return dtc::Event::DEFAULT;
  }
);

auto BA = G.stream(B, A);
```

Stream A→B

```
A.on(
  [&AB] (dtc::Vertex& v) {
    (*v.ostream(AB))("hello world from A"s);
    dtc::cout("Sent 'hello world from A' to stream ", AB, "\n");
  }
);

G.container().add(B).cpu(1).memory(1_GB);

B.on(
  [&BA] (dtc::Vertex& v) {
    (*v.ostream(BA))("hello world from B"s);
    dtc::cout("Sent 'hello world from B' to stream ", BA, "\n");
  }
);

BA.on(
  [] (dtc::Vertex& A, dtc::InputStream& is) {
    if(std::string a; is(a) != -1) {
      dtc::cout("Received: ", a, "\n");
      return dtc::Event::REMOVE;
    }
    return dtc::Event::DEFAULT;
  }
);
```

Stream B→A

```
dtc::Executor(G).run();
```

- **Only a couple lines of code**
- Single sequential program
- Distributed across computers
- No explicit data management
- Easy-to-use streaming interface
- Asynchronous by default
- Scalable to many threads
- Scalable to many machines
- In-context resource controls
- Scale out to heterogeneous devices
- Transparent concurrency controls
- Robust runtime via Linux container

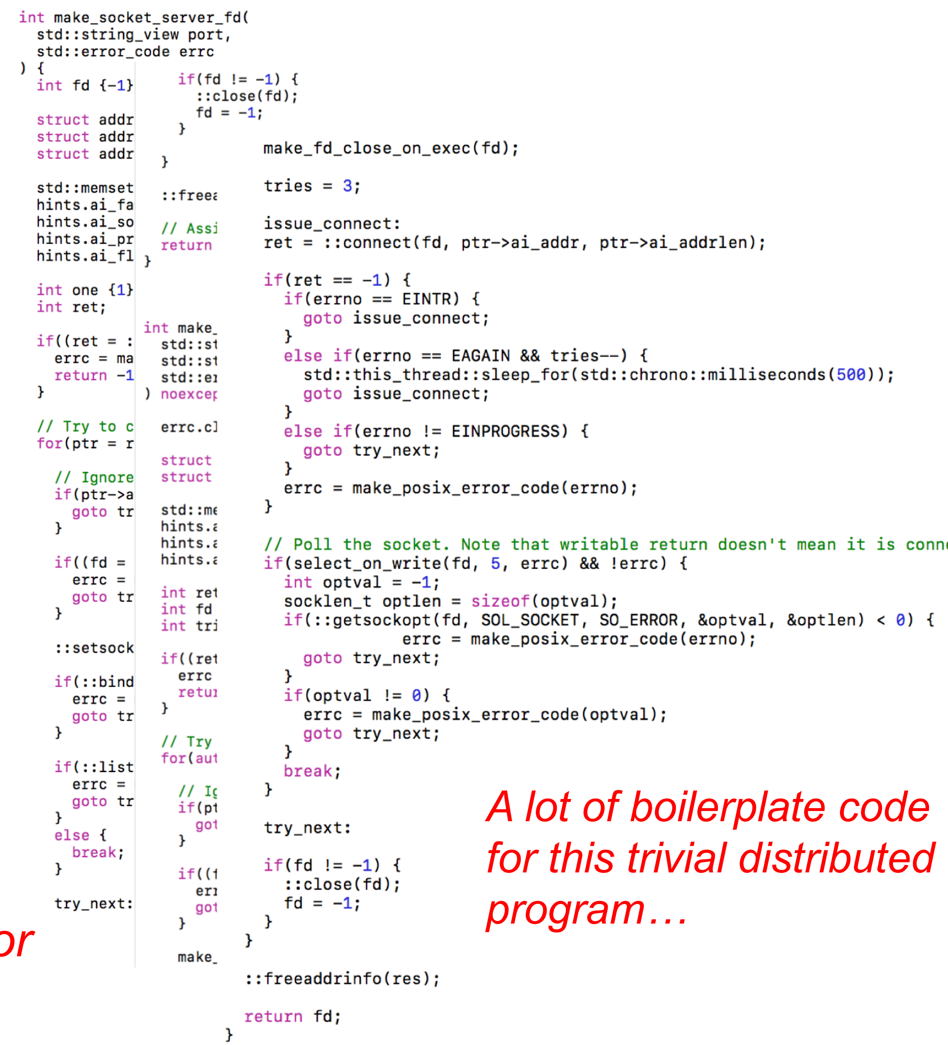
… and more

Distributed Hello-world without DtCraft …

```cpp
auto count_A = 0;
auto count_B = 0;

// Send a random binary data to fd and add the
// received data to the counter.
auto pinpong(int fd, int& count) {
  auto data = random<bool>()
  auto w = write(fd, &data, sizeof(data));
  if(w == -1 && errno != EAGAIN) {
    throw system_error("Failed on write");
  }
  data = 0;
  auto r = read(fds, &data, sizeof(data));
  if(r == -1 && errno != EAGAIN) {
    throw system_error("Failed on read");
  }
  count += data;
}

int fd = -1;
std::error_code errc;

if(getenv("MODE") == "SERVER") {
  fd = make_socket_server_fd("9999", errc);
}
else {
  fd = make_socket_client_fd("127.0.0.1", "9999", errc);
}

if(fd == -1) {
  throw system_error("Failed to make socket");
}
```

server.cpp

client.cpp



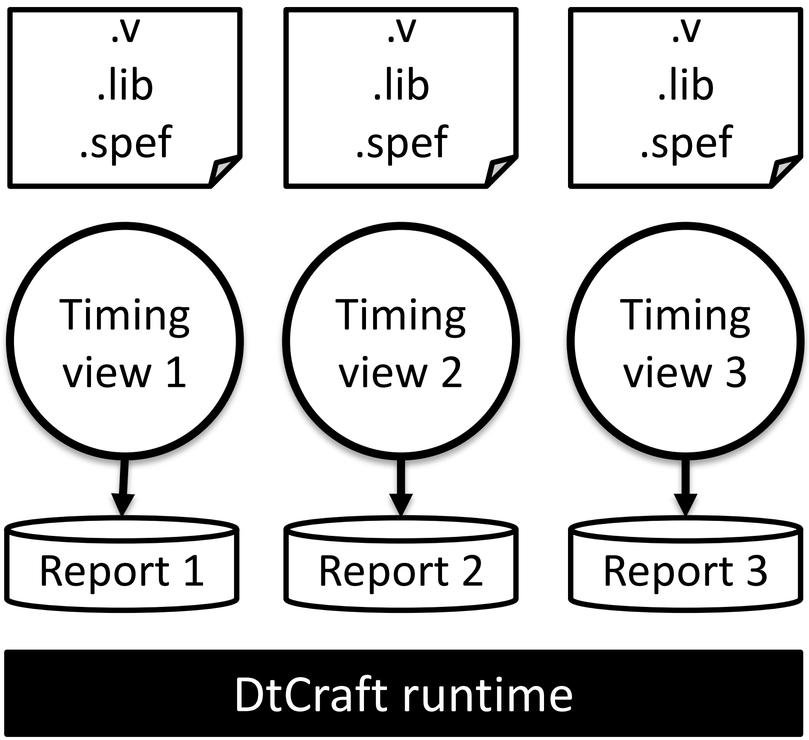
A lot of boilerplate code for this trivial distributed program…

Branch your code to server and client for distributed computation!

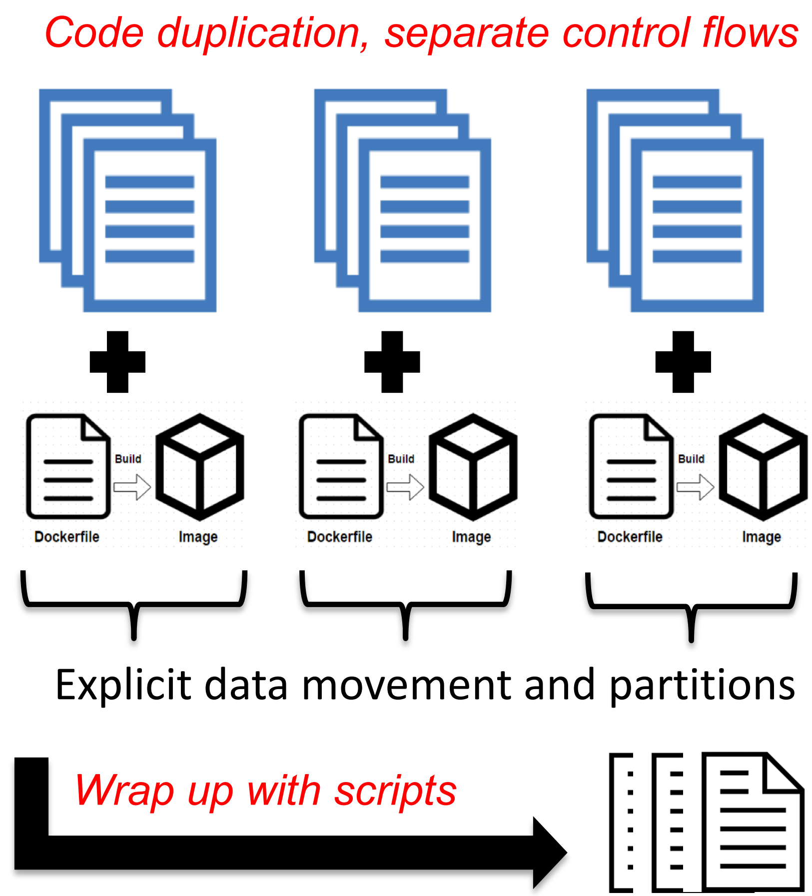
simple.cpp → server.cpp + client.cpp

Distributed Timing with DtCraft

❑ Three timing views



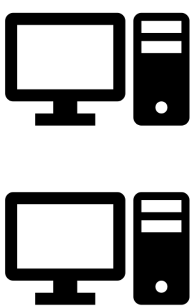
With DtCraft (<100 lines of code)



Without DtCraft (hard-coded)

Demo (dta.cpp)

- ❑ **Distributed timing analysis with three timing views**
  - ❑ simple_tv1 (P=1, V=0.7, T=70)
  - ❑ simple_tv2 (P=0.5, V=0.95, T=85)
  - ❑ simple_tv3 (P=0.9, V=0.5, T=60)
- ❑ **OpenTimer how-to (by Kunal)**
  - ❑ https://www.udemy.com/vlsi-academy-sta-checks-2/
- ❑ **Local mode execution**
- ❑ **Distributed mode execution**



Login user	host	DtCraft home
twhuang	csl-408-08.csl.Illinois.edu	/home/twhuang/DtCraft
twhuang	csl-408-14.csl.Illinois.edu	/home/twhuang/DtCraft

Notice: Replace with your own login/hosts/DtCraftHome.

Demo code: http://web.engr.illinois.edu/~thuang19/webinar.tar.gz

Debrief

- **Transparency**
  - No low-level network programming details
  - Automatic workload distribution
- **Scalability**
  - Same code scales out automatically when new machines added
  - Dynamic scaling
  - Flexible partitions and in-context resource controls
- **Programmability**
  - Can incorporate other programs together
- **Productivity**
  - Less than 100 lines of code

Exercise (lab.cpp)

- **Distributed timing for two designs each with four views**
  - c17_v1, c17_v2, c17_v3, c17_v4
  - c499_v1, c499_v2, c499_v3, c499_v4
- **Implement a stream graph**
  - Eight vertices each operating on one timing view
  - Two containers, one for c17_v* and another for c499_v*
  - Try different resource assignments for each container
- **Submit the graph to your DtCraft cluster**
  - Local mode and distributed mode
- **Report TNS and WNS for each view**
- **Report elapsed time and peak memory for each container**
- **Use at most 50 lines of code ☺**

Example Solution (20 lines)

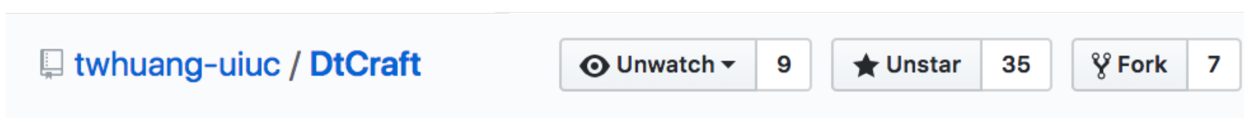
```
#include <dtc/dtc.hpp>

int main(int argc, char* argv[]) {
  using namespace dtc::literals;

  dtc::Graph G;
  auto c17tv1 = G.vertex().program("path_to_webinar/ot.sh path_to_webinar/c17_tv1 tv1.report");
  auto c17tv2 = G.vertex().program("path_to_webinar/ot.sh path_to_webinar/c17_tv2 tv2.report");
  auto c17tv3 = G.vertex().program("path_to_webinar/ot.sh path_to_webinar/c17_tv3 tv3.report");
  auto c17tv4 = G.vertex().program("path_to_webinar/ot.sh path_to_webinar/c17_tv4 tv4.report");

  auto c499tv1 = G.vertex().program("path_to_webinar/ot.sh path_to_webinar/c499_tv1 tv1.report");
  auto c499tv2 = G.vertex().program("path_to_webinar/ot.sh path_to_webinar/c499_tv2 tv2.report");
  auto c499tv3 = G.vertex().program("path_to_webinar/ot.sh path_to_webinar/c499_tv3 tv3.report");
  auto c499tv4 = G.vertex().program("path_to_webinar/ot.sh path_to_webinar/c499_tv4 tv4.report");

  G.container().add(c17tv1).add(c17tv2).add(c17tv3).add(c17tv4).cpu(1).memory(1_GB);
  G.container().add(c499tv1).add(c499tv2).add(c499tv3).add(c499tv4).cpu(1).memory(1_GB);

  dtc::Executor(G).run();
  return 0;
}
```

Example Report

Design	TNS (ps)	WNS (ps)	Elapsed Time	Peak Memory
c17_tv1	-8.14469e+02	-2.29314e+01	0.23s	86M
c17_tv2	-2.79547e+03	-7.32566e+01		
c17_tv3	-7.15888e+02	-1.91890e+01		
c17_tv4	-1.69670+e03	-4.53052e+01		
c499_tv1	-4.97395e+05	-5.16786e+02	0.25s	90M
c499_tv2	-4.86524e+05	-5.05552e+02		
c499_tv3	-2.34709e+05	-2.44073e+02		
c499_tv4	-1.69093e+06	-1.75538e+03		

Conclusion

- ❑ **Express your parallelism in the right way**
  - ❑ A “hard-coded” distributed timing analysis framework
- ❑ **Boost your productivity in writing parallel code**
  - ❑ DtCraft system
- ❑ **Leverage your time to produce promising results**
  - ❑ Demo 1: A vanilla example
  - ❑ Demo 2: Distributed timing using DtCraft
  - ❑ Lab



Please star and watch DtCraft at GitHub to receive updates!
(https://github.com/twhuang-uiuc/DtCraft)

Thank you!

Dr. Tsung-Wei Huang

twh760812@gmail.com

Github: https://github.com/twhuang-uiuc

Website: http://web.engr.illinois.edu/~thuang19/