Catamaran/ARIFIC:
Architecture Research in FPGAs in the Cloud

Tutorial at HPCA-29, Montreal

Rishiyur S. Nikhil
Bluespec, Inc.
Sunday, February 26, 2023

GitHub repo for this tutorial: https://github.com/rsnikhil/Tutorial_at_HPCA-29

What is Catamaran/ARIFIC?

Architecture Researcher's Focus

- CPU microarchitecture
- Memory systems (caches, MMUs, coherence, WMMs, …)
- Accelerators

RISC-V CPU	...	RISC-V CPU	Tightly-coupled accelerator	Loosely-coupled accelerator

Memory subsystem (caches, MMUs, …)

Catamaran/ARIFIC (infrastructure in the cloud)

Host computer
running Linux

System Control:
- Load ELF/memhex (programs, data)
- into DDR
- Assert/Deassert Core's RESET
- Configure CPU
- Collect stats, traces

Console (to/from UART)

Disk-device support

Network-device support

GDB control of CPU on FPGA

PCIe connection

FPGA

UART (for console)

Disk-device support

Network-device support

Support for counters, traces, etc.

(substitutable RISC-V core)

DDR memory

What are "FPGAs in the cloud"?

ARIFIC (infrastructure in the cloud)

Host computer

System Control:
- Load ELF/memhex (programs, data) into DDR
- Assert/Deassert Core's RESET
- Configure CPU
- Collect stats, traces

Console (to/from UART)

Disk-device support

Network-device support

GDB control of CPU on FPGA

PCIe connection

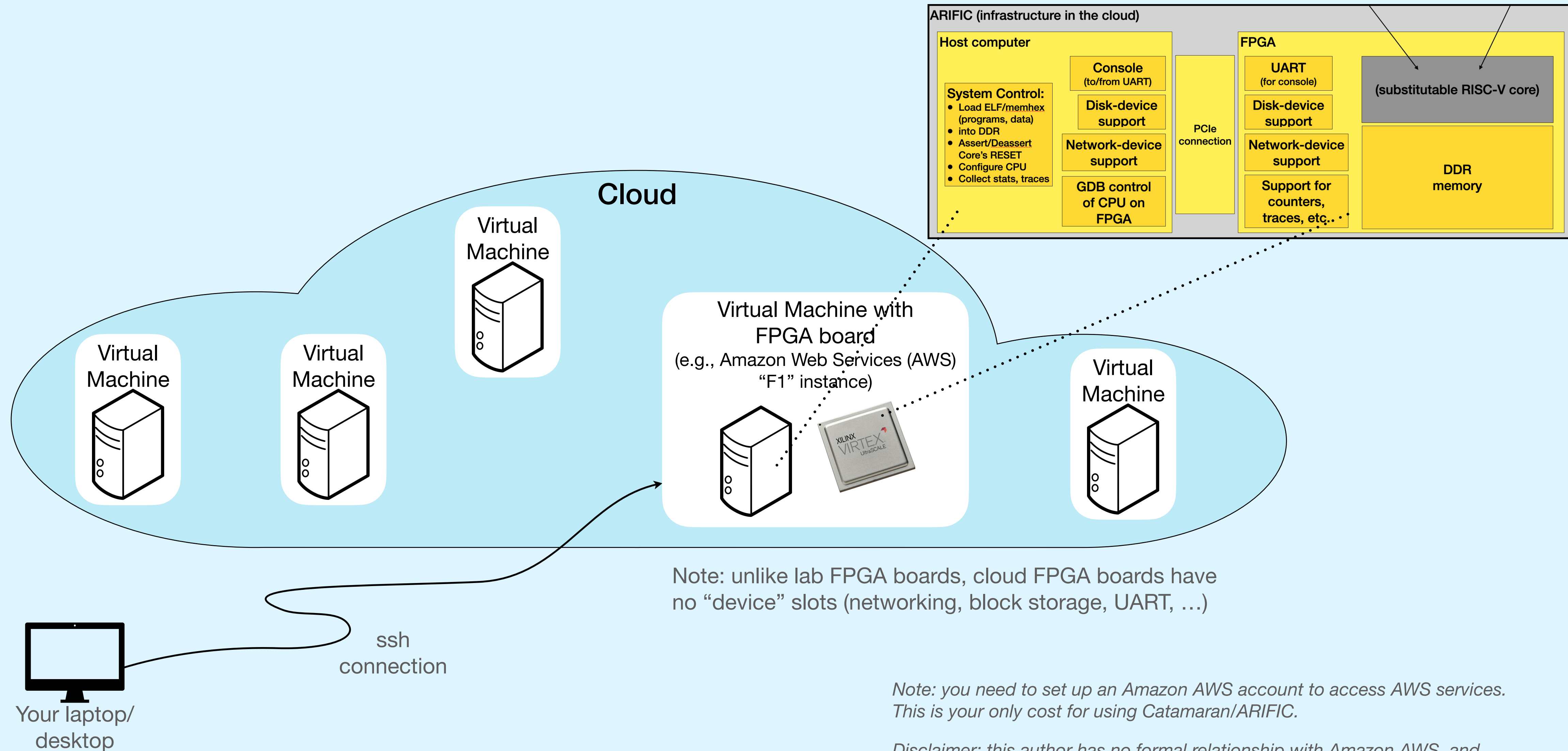**FPGA**

UART (for console)

Disk-device support

Network-device support

Support for counters, traces, etc..

(substitutable RISC-V core)

DDR memory

Cloud

Virtual Machine

Virtual Machine

Virtual Machine

Virtual Machine with FPGA board
(e.g., Amazon Web Services (AWS) "F1" instance)

XILINX VIRTEX UltraSCALE

Virtual Machine

Note: unlike lab FPGA boards, cloud FPGA boards have no "device" slots (networking, block storage, UART, …)

ssh connection

Your laptop/ desktop

3

Note: you need to set up an Amazon AWS account to access AWS services. This is your only cost for using Catamaran/ARIFIC.

Disclaimer: this author has no formal relationship with Amazon AWS, and is merely an ordinary user of their services.

Plan for this tutorial

- Basics of Amazon AWS virtual machines

- Demo/description of what you can do with Catamaran/ARIFIC

  - Run ISA tests

  - Cross-compile and run bare-metal (no OS) C programs

  - Run Linux, with networking and block devices

  - Cross-compile and run C programs under Linux

- Demo/description of plugging in your own RISC-V Core[1]

  - Standard RTL-level interface for the core

  - Build a full-system simulation (using Verilator), and run it
    (can do on your local computer; does not need cloud)

  - Build a bitfile for AWS F1 FPGA, and run it

- Use GDB to control the RISC-V CPU on the FPGA

[1] "Your own RISC-V Core"
Options:
- Your own new CPU design
- Modify available CPU (many open-source)
  - microarchitecture change
  - new instruction
  - new CSRs (e.g., counters)
- Modify/replace memory system (caches, MMUs, PMPs, PTWs, …)
- Add tightly-coupled or loosely coupled accelerator
- … or other research idea …

Basics of Amazon AWS

- *AMIs (virtual machines in the Amazon cloud)*

- Connecting to an AMI using "ssh"

- About AWS bitfiles

Basics of Amazon AWS
AMIs (virtual machines in Amazon cloud)

Cloud

Virtual
Machine

Virtual
Machine

• • •

Virtual
Machine

ssh connection,
scp copy, …

Your laptop/
desktop
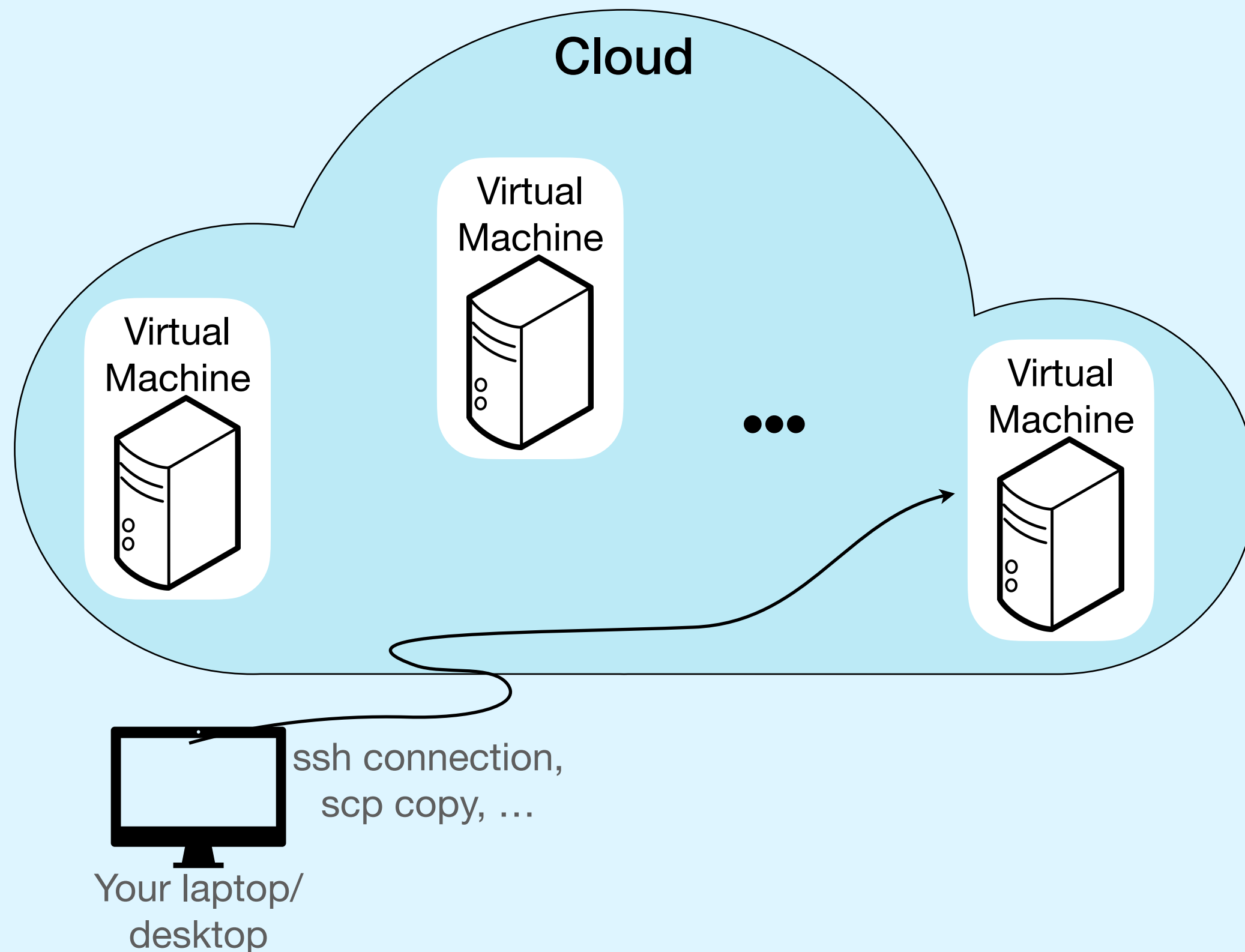
Amazon terminology for a virtual machine in their cloud:
"AMI" = "Amazon Machine Instance"
We also just say "instance" for an AMI/virtual machine

With an Amazon AWS account, you can "create" one or more AMIs for yourself (see Appendix for info on how to create an AMI).

For Catamaran/ARIFIC we usually create two AMIs:

• For **building** FPGA bitfiles: choose the free "FPGA Developer AMI" from AWS Marketplace, which is CentOS + Xilinx tools (Vivado) and licenses + aws-fpga SDK and HDK, running on "m6i.2xlarge" instance type.

• For **running** Catamaran/ARIFIC on FPGA: choose the free basic standard Ubuntu 22.04 AMI from AWS Marketplace, running on an "f1.2xlarge" instance type. "f1" instances have FPGAs attached.

Costs:
• For using Catamaran/ARIFIC, your only costs are AWS costs (between you and Amazon)
• AWS charges vary by instance type. Charges for "f1" instances (with FPGA) are somewhat higher, hence our choice above of a cheaper instance for our "build" AMI, which does not need an attached FPGA.
• *Hint*: put your AMI into the "stopped" state when not in use, to save charges.

Basics of Amazon AWS

AMIs (virtual machines in Amazon cloud)

Cloud

Virtual Machine

Virtual Machine

Virtual Machine

ssh connection, scp copy, …

Your laptop/ desktop
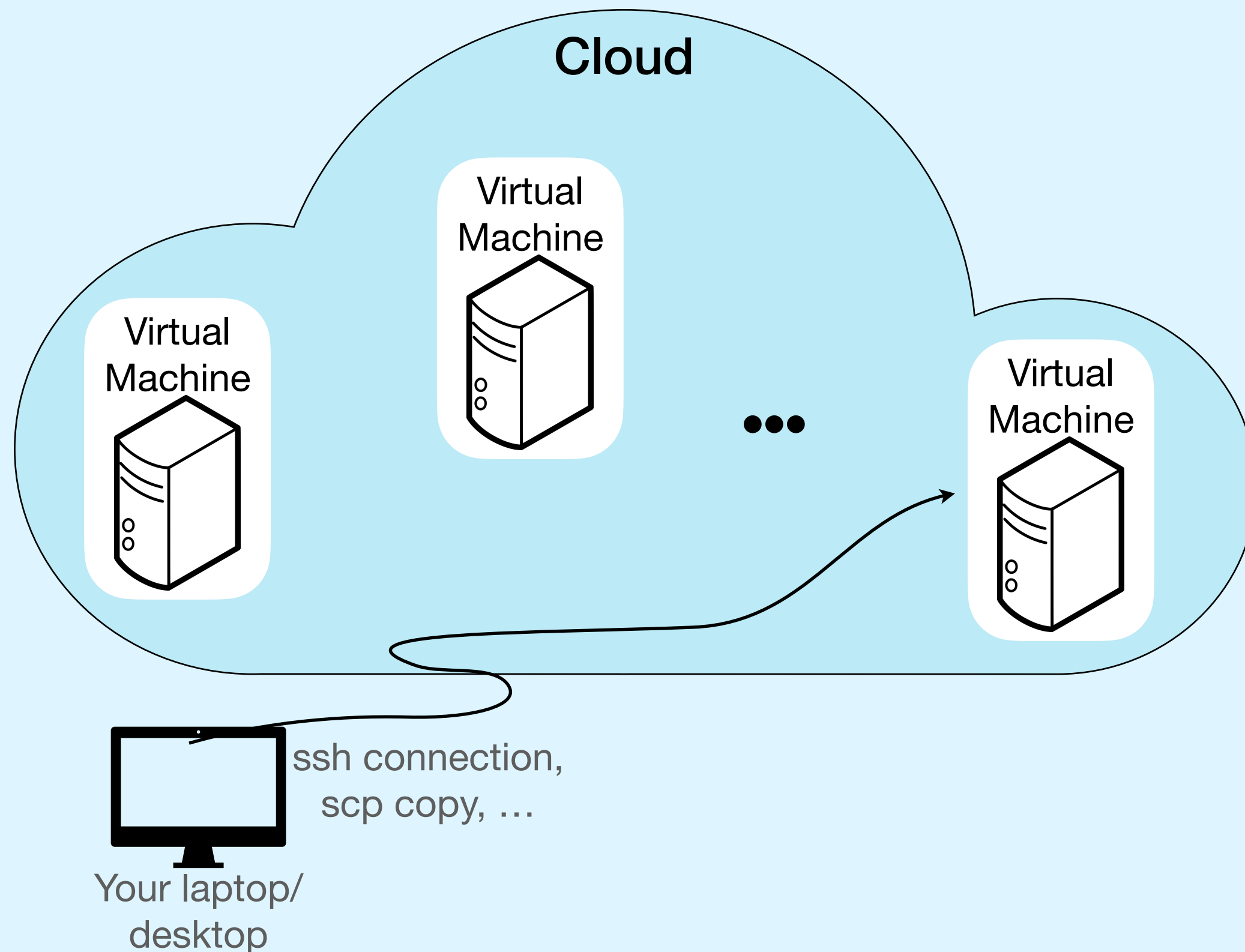
Please see Appendix for how to install:

- aws-fpga SDK and HDK
  - Needed to go from RTL to bitfiles
  - Needed for installing Xilinx XDMA driver for host-FPGA PCIe communication
- CLI (command-line interface)
  - Needed for loading bitfiles into FPGA, etc.

Basics of Amazon AWS

- AMIs (virtual machines in the Amazon cloud)

- *DEMO: Connecting to an AMI using "ssh"*

- About AWS bitfiles

Basics of Amazon AWS
Connecting to your AMI with ssh   (1/2)

In "EC2 Instances" dashboard …

Use "Instance state" drop-down menu to start/stop selected instance

Select your instance

Make sure your AMI is "Running"

Note the IPv4 DNS address for the instance.
Hint: click to copy to clipboard

Basics of Amazon AWS
Connecting to your AMI with ssh   (2/2)

In a terminal on your laptop/desktop, connect to your AMI:

```
$ ssh -i ~/.ssh/MyPrivateKey.pem  ubuntu@ec2-54-173-149-133.compute-1.amazonaws.com
```

Note: you can copy files to and from your AMI using "scp" (which is based on "ssh")

```
$ scp -i ~/.ssh/MyPrivateKey.pem  \
         localfile \
         ubuntu@ec2-54-173-149-133.compute-1.amazonaws.com:~/remotefile
```

```
$ scp -i ~/.ssh/MyPrivateKey.pem  \
         ubuntu@ec2-54-173-149-133.compute-1.amazonaws.com:~/remotefile \
         localfile
```

Basics of Amazon AWS
DEMO: connect to your "run" AMI
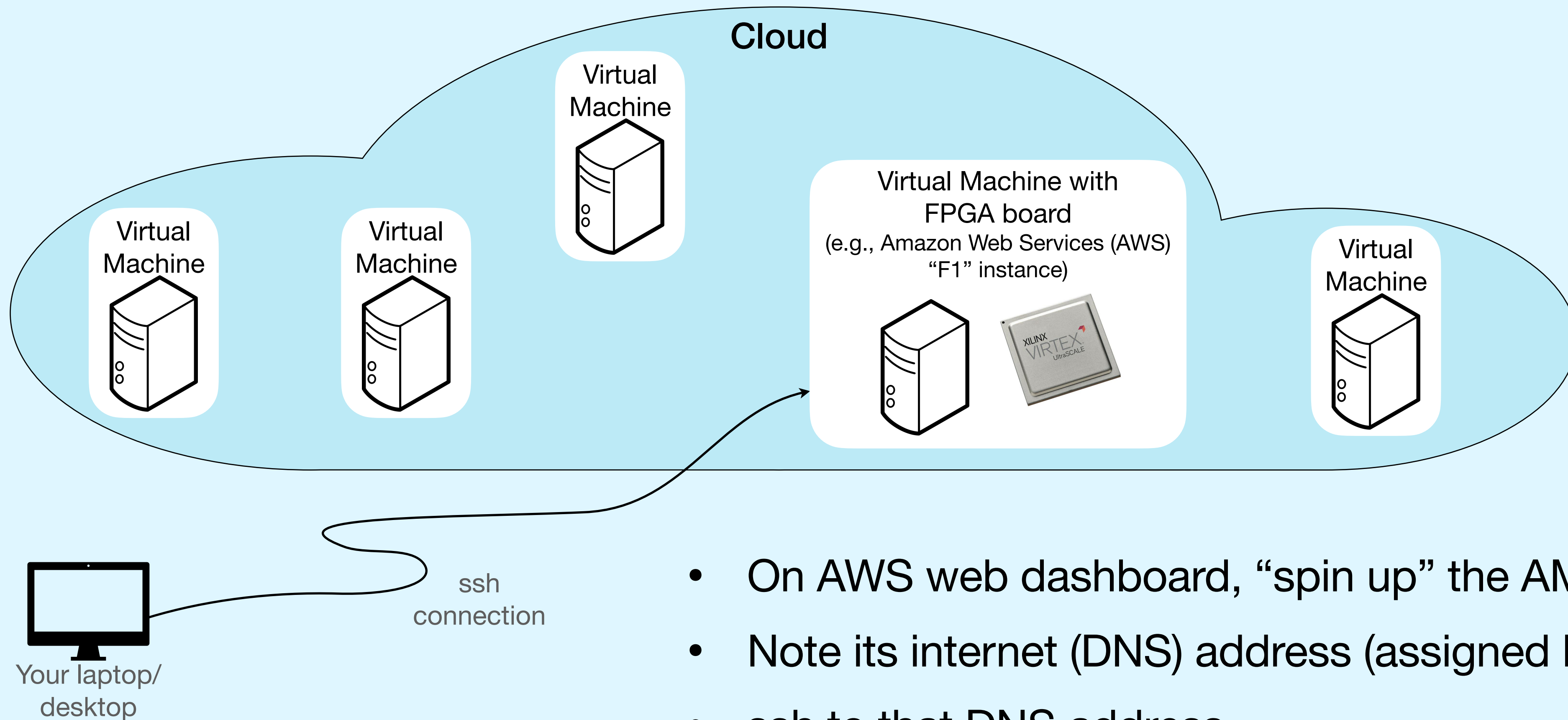
- On AWS web dashboard, "spin up" the AMI into "running" state
- Note its internet (DNS) address (assigned by AWS during spin-up)
- ssh to that DNS address
- Tour of tutorial directory contents

Basics of Amazon AWS

- AMIs (virtual machines in the Amazon cloud)

- DEMO: Connecting to an AMI using "ssh"

- *About AWS bitfiles*

Basics of Amazon AWS
About AWS bitfiles

- Unlike "on-premises" FPGAs (on your lab bench/desktop), on AWS, FPGA bitfiles reside somewhere in the cloud (we have no direct access to the actual bitfile).

- Each bitfile has unique ids:

  - AFI (Amazon FPGA Image).   These are unique within an AWS geographic region.

  - AGFI (…Global…).   These are unique across all AWS geographic regions.

- See later section of this tutorial re. creating a bitfile and obtaining its AFI and AGFI

- AWS makes available software that, given an AGFI, will fetch the bitfile from somewhere in the cloud and load it into the FPGA in your instance.

(The author is unclear why AWS has two IDs for a bitfile, the AFI and AGFI, instead of just the AGFI?)

What you can do with Catamaran/ARIFIC
Available pre-built bitfiles, all based on open-source CPUs

CPU	Original HDL	ISA	Runs bare-metal programs	Runs Linux
Berkeley/SiFive Rocket	Chisel	RV64GC MSU privilege levels Sv39 virtual memory	Yes	Yes
Bluespec Flute	BSV	RV64GC MSU privilege levels Sv39 virtual memory	Yes	Yes
OpenHardware Group CVA6 (a.k.a. ETH Zurich Ariane)	SystemVerilog	RV64GC MSU privilege levels Sv39 virtual memory	Yes	Yes [1]

In this tutorial we'll show demos on Amazon AWS with Rocket and CVA6

[1] Development in progress; expected March 2023

Basics of Amazon AWS
DEMO: Loading an AWS bitfile into the FPGA
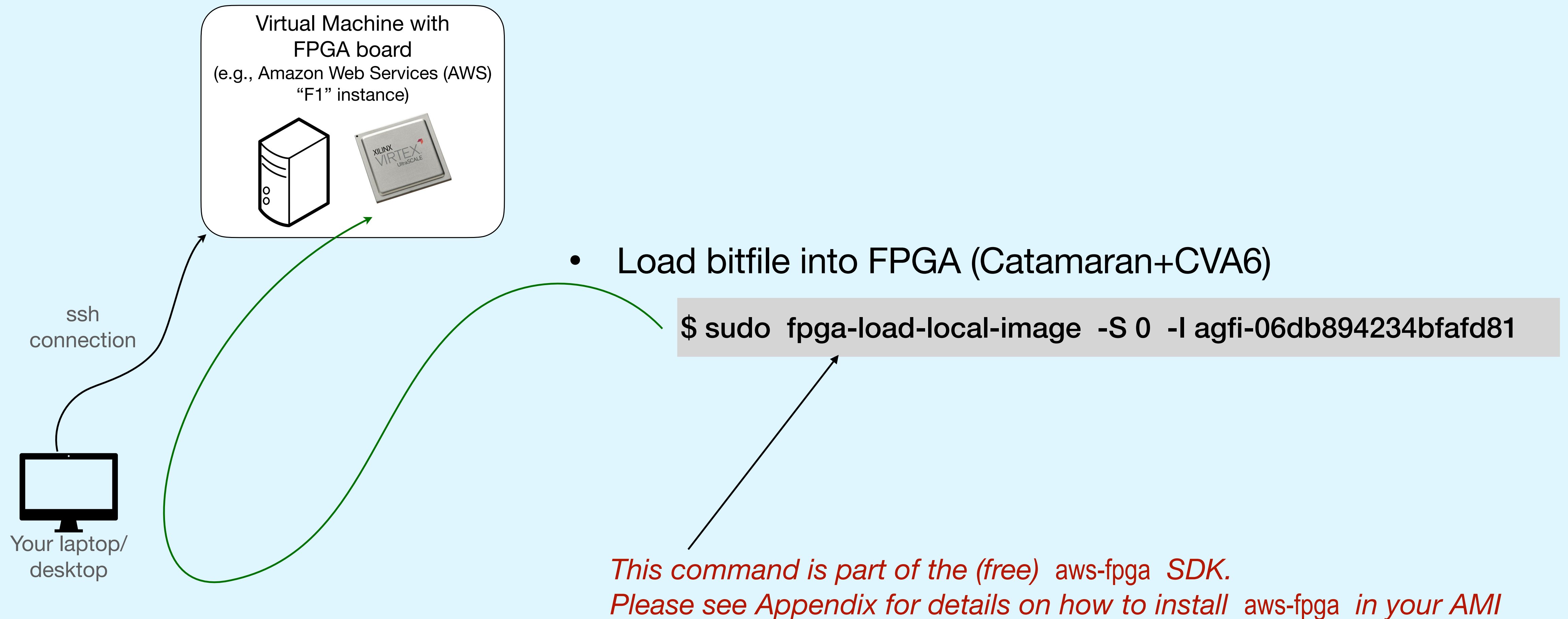
Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

ssh
connection

Your laptop/
desktop

- Load bitfile into FPGA (Catamaran+CVA6)

$ sudo fpga-load-local-image -S 0 -I agfi-06db894234bfafd81

This command is part of the (free) aws-fpga SDK.
Please see Appendix for details on how to install aws-fpga in your AMI

Demo/description of what you can do with Catamaran/ARIFIC

- *DEMO: Cross-compile and run ISA tests*

- DEMO: Cross-compile and run bare-metal (no OS) C programs

- DEMO: Run Linux, with networking and block devices

- DEMO: Cross-compile and run C programs under Linux

What you can do with Catamaran/ARIFIC
Run ISA tests

What are "ISA tests"?

- RVI (RISC-V International, https://riscv.org) provides a standard set of tests for the RISC-V Instruction Set Architecture (ISA)

  - https://github.com/riscv-software-src/riscv-tests

- Consists of a number of assembly-language program files; each one focuses on testing one particular RISC-V instruction (opcode)

  - 236 tests for RV64 IMAFDC + MSU privileges + Sv39 Virtual Memory
  - 173 tests for RV32 IMAFDC + MSU privileges + Sv32 Virtual Memory

- Each program contains a number of sub-tests (each self-checking)

- Each program ends by writing to a particular "tohost" MMIO address

  - 0, if all sub-tests succeed (pass)
  - N, if sub-test N failed

What you can do with Catamaran/ARIFIC

Run ISA tests

Cross-compiling the RISC-V ISA tests

Please see Appendix for info on installing the (free) RISC-V GNU Toolchain ("gcc" and friends).

This tutorial repository contains, in the dir ISA_Tests/

- A copy of the RISC-V ISA tests source code in ISA_Tests/isa/
  (see GitHub URL on previous slide)

- A Makefile and linker scripts to cross-compile the RISC-V ISA tests for RV64GC+MSU+Sv39

- Pre-built results of cross-compiling: set of RISC-V ELF files for the tests
  in ISA_Tests/isa/elfs

Note: our linker script sets
- Catamaran/ARIFIC program-start address = 0x_8000_0000
- "tohost" MMIO address = 0x_6fff_0010
- See ISA_Tests/README.txt for details

Run ISA tests
DEMO: cross-compile RISC-V ISA tests
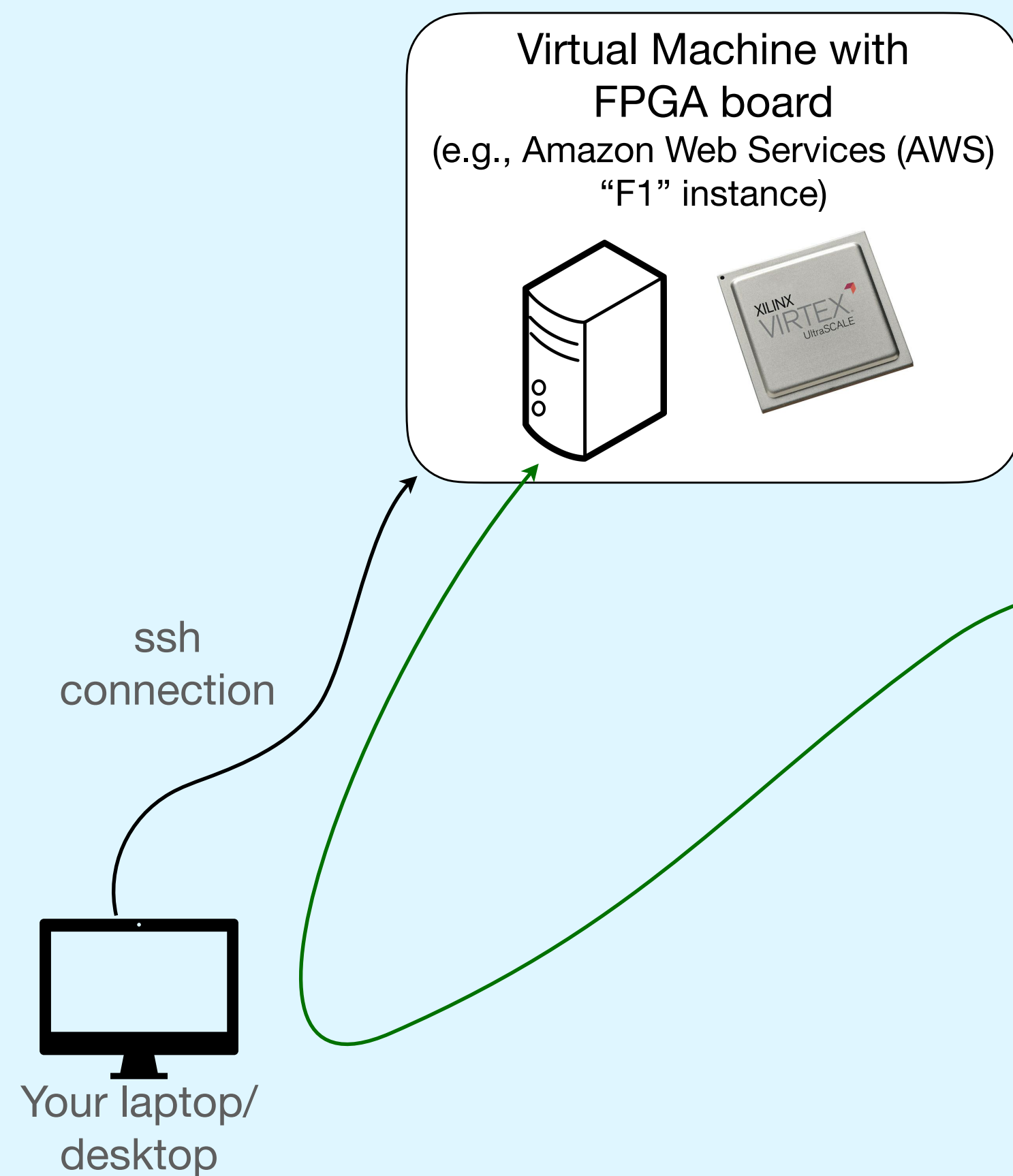
Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

ssh
connection

Your laptop/
desktop

Prerequisites:
- your RISC-V Gnu Toolchain (including cross-compiler gcc) is installed
- you've defined RISCV environment variable to point at your toolchain installation directory
- you've added ${RISCV}/bin to your PATH environment variable.

```
$ cd  ISA_Tests/isa
$ make
```

It will produce 100s of ELF files with a naming convention like this:

    rv64ui-p-add   (user-level, integer, physical memory, 'add' opcode)
    rv64ui-v-add   (…, virtual memory, …)

as well as "object dump" (disassembly) files likes this:

    rv64ui-p-add.dump

Run ISA tests
DEMO: run RISC-V ISA tests
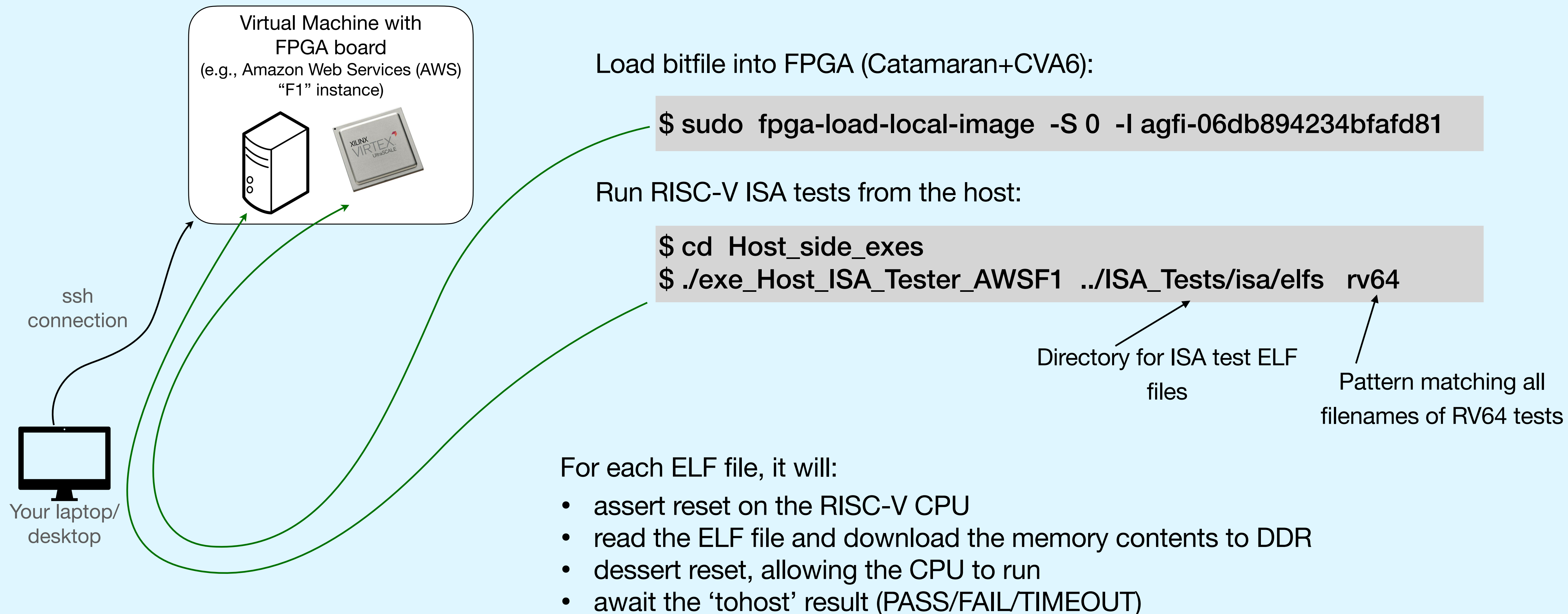
Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

ssh
connection

Your laptop/
desktop

Load bitfile into FPGA (Catamaran+CVA6):

```
$ sudo  fpga-load-local-image  -S 0  -I agfi-06db894234bfafd81
```

Run RISC-V ISA tests from the host:

```
$ cd  Host_side_exes
$ ./exe_Host_ISA_Tester_AWSF1  ../ISA_Tests/isa/elfs  rv64
```

Directory for ISA test ELF
files

Pattern matching all
filenames of RV64 tests

For each ELF file, it will:
• assert reset on the RISC-V CPU
• read the ELF file and download the memory contents to DDR
• dessert reset, allowing the CPU to run
• await the 'tohost' result (PASS/FAIL/TIMEOUT)

Demo/description of what you can do with Catamaran/ARIFIC

- DEMO: Cross-compile and run ISA tests

- *DEMO: Cross-compile and run bare-metal (no OS) C programs*

- DEMO: Run Linux, with networking and block devices

- DEMO: Cross-compile and run C programs under Linux

Cross-compile and run bare-metal (no OS) C programs

What are "bare-metal" C programs?

- No system calls. "printf/putc/putchar", "scanf/getc/getchar" are linked with libraries that directly interact with the UART (device that sends/receives character to terminal)

- No "exit"; typically end in an infinite idle loop
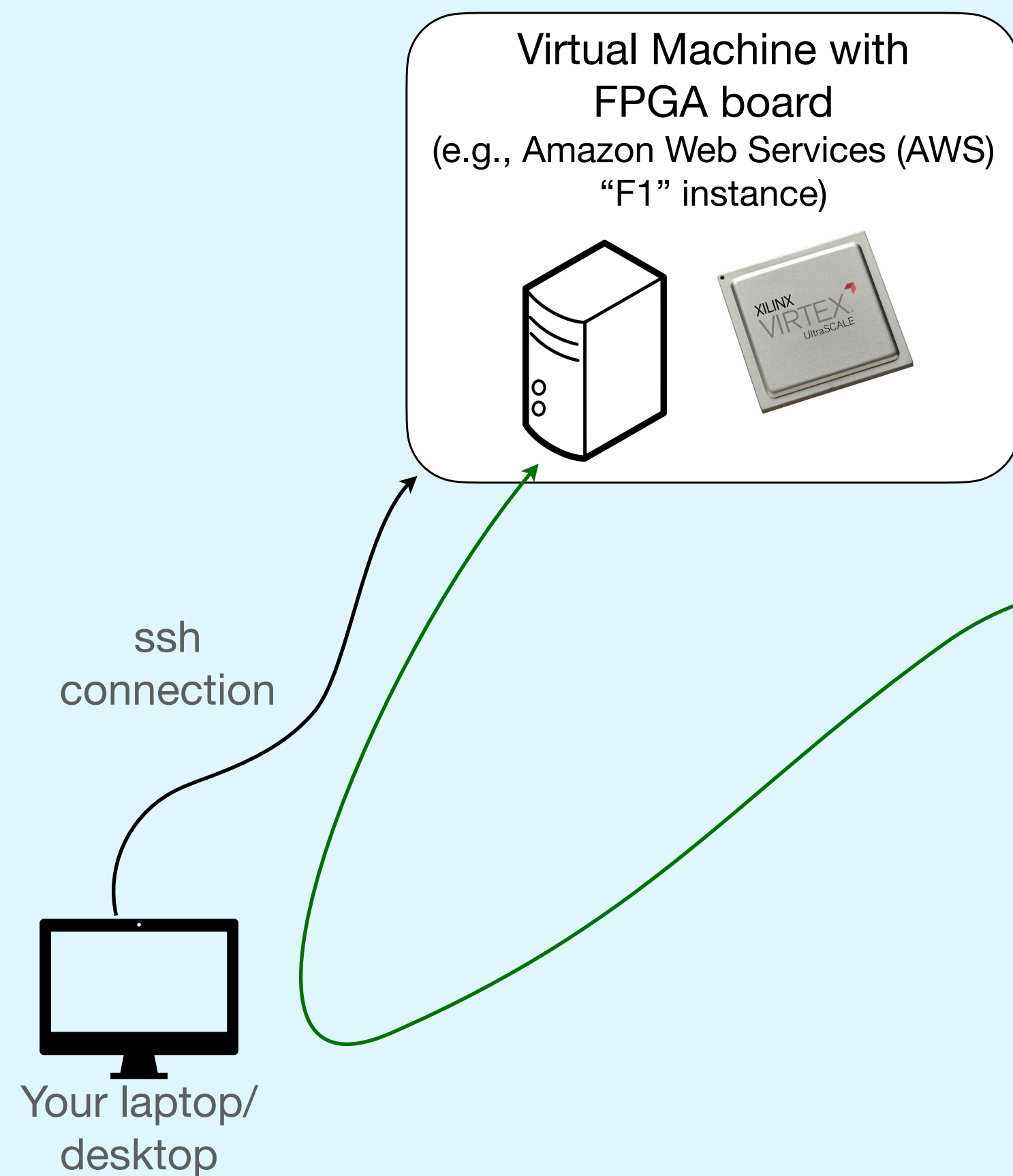
This tutorial repository contains

- Two small C programs: "Hello World!" and "cat" (echoes stdin to stdout)

- A Makefile and linker scripts to cross-compile the C programs for RV64GC

- Pre-built results of cross-compiling the examples: two RISC-V ELF files

Note: our linker script sets
- Catamaran/ARIFIC program-start address = 0x_8000_0000
- UART MMIO address = 0x_6010_0000

Run ISA tests
DEMO: cross-compile bare-metal C programs

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

ssh
connection

Your laptop/
desktop

Prerequisites:
- your RISC-V Gnu Toolchain (including cross-compiler gcc) is installed
- you've defined RISCV environment variable to point at your toolchain installation directory
- you've added ${RISCV}/bin to your PATH environment variable.

Classical "Hello World!" program
```
$ cd  C_Examples/hello
$ make  all_bare
```
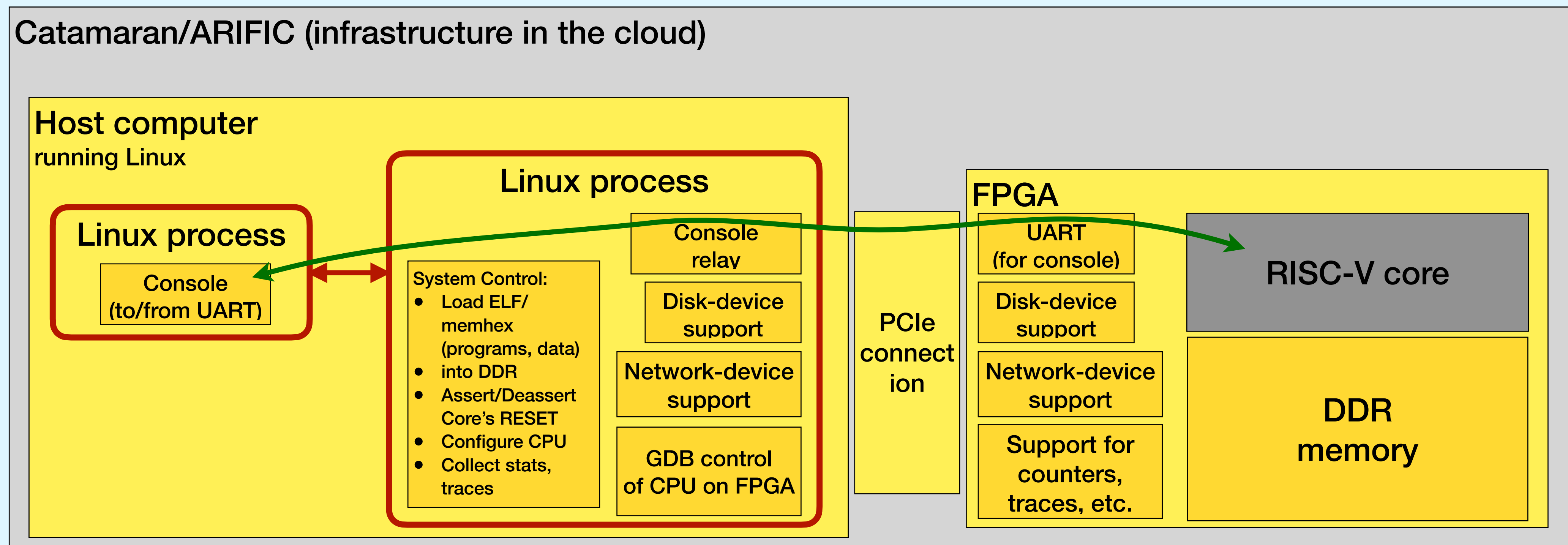
Creates: hello.RV64.bare.elf hello.RV64.bare.map hello.RV64.bare.objdump

Program that just echoes stdin to stdout (UART-in to UART-out)
```
$ cd  C_Examples/cat
$ make  all_bare
```
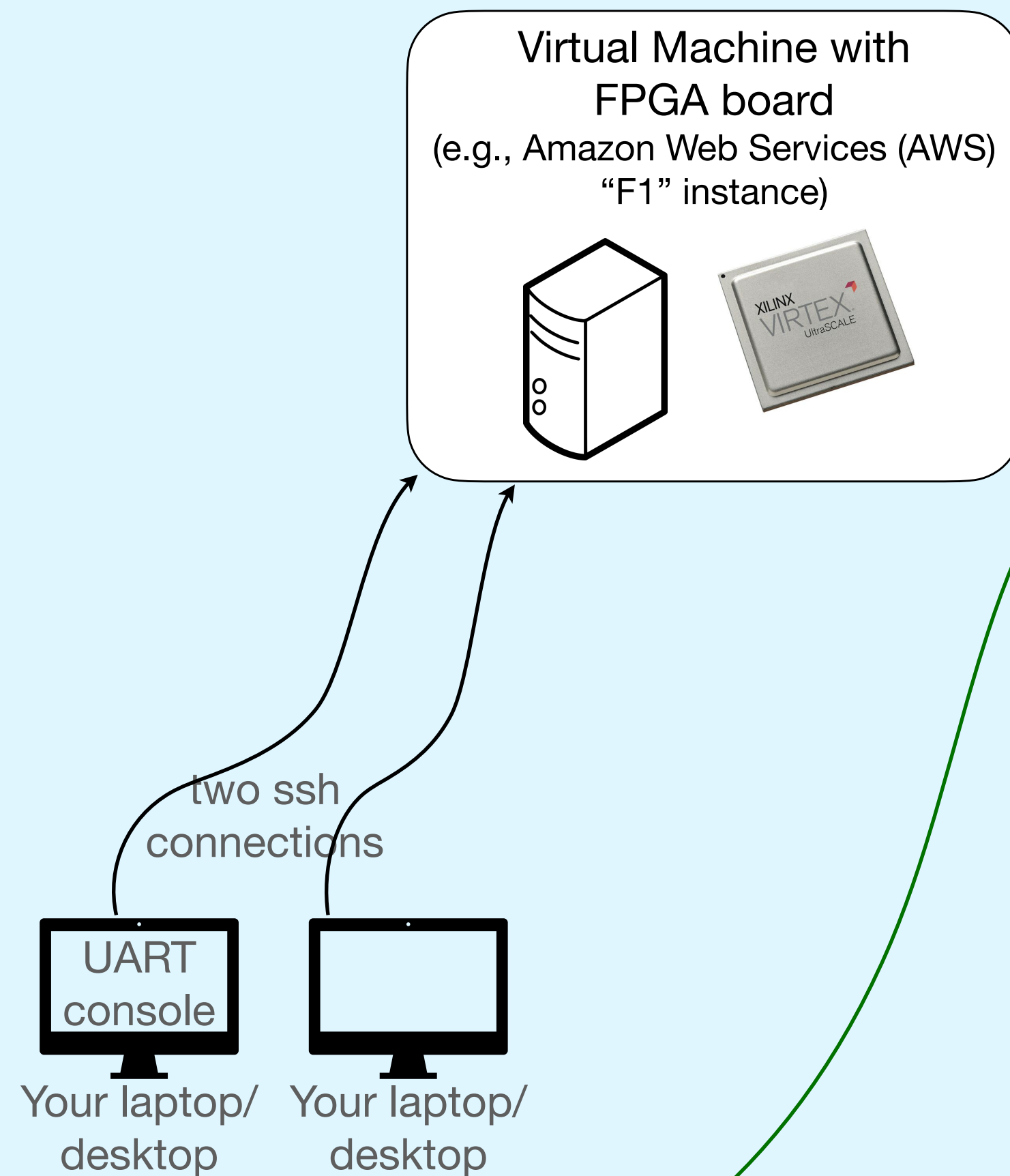
Cross-compile and run bare-metal (no OS) C programs

- Bare-metal C programs have no system calls. "printf/putc/putchar", "scanf/getc/getchar" are linked with libraries that directly interact with the UART (hardware device that sends/receives characters)

- Catamaran/ARIFIC connects the UART to a terminal on the host computer

Catamaran/ARIFIC (infrastructure in the cloud)

Host computer
running Linux

Linux process

Linux process

Console
(to/from UART)

System Control:
- Load ELF/ memhex (programs, data) into DDR
- Assert/Deassert Core's RESET
- Configure CPU
- Collect stats, traces

Console relay

Disk-device support

Network-device support

GDB control of CPU on FPGA

PCIe connection

FPGA

UART (for console)

Disk-device support

Network-device support

Support for counters, traces, etc.

RISC-V core

DDR memory

Cross-compile and run bare-metal (no OS) C programs
DEMO: Open a second terminal on the AMI for the UART process

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

XILINX
VIRTEX
UltraSCALE

two ssh
connections

UART
console

Your laptop/
desktop

Your laptop/
desktop

Make 2nd ssh connection to AMI; start UART console

```
$ ssh -i ~/.ssh/MyPrivateKey.pem  \
        ubuntu@ec2-54-173-149-133.compute-1.amazonaws.com
```
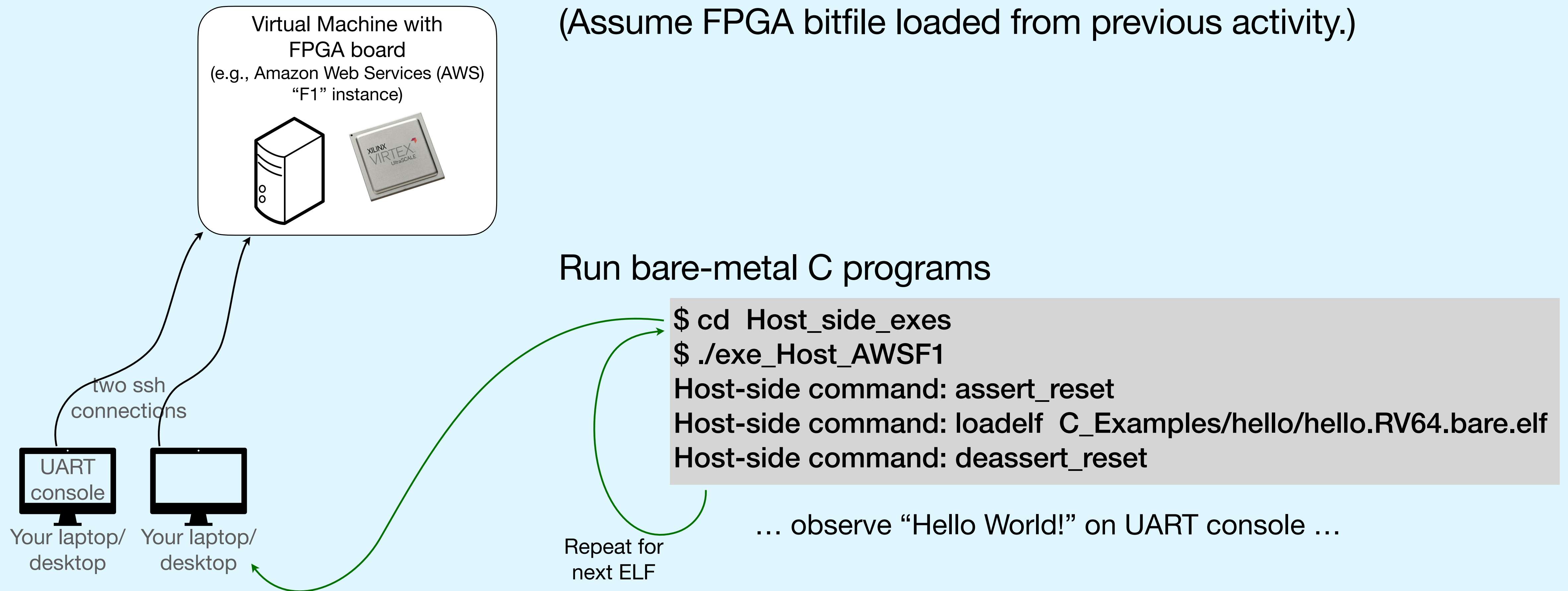
In the terminal, start the "UART_Console" program

```
$ sudo UART_Console  —log_UART.txt
```

```
--22:32:22--ip-172-31-83-224: ~/Git/Tutorial_at_HPCA-29/Host_side_exes
$ sudo ./UART_Console --log_UART.txt
ptyname_file filename = UART_ptyname_file
UART output logfile name = (null)
Awaiting file 'UART_ptyname_file' for reading pty name
```

Cross-compile and run bare-metal (no OS) C programs

DEMO: run bare-metal C programs

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

(Assume FPGA bitfile loaded from previous activity.)

two ssh
connections

UART
console

Your laptop/
desktop

Your laptop/
desktop

Run bare-metal C programs

```
$ cd  Host_side_exes
$ ./exe_Host_AWSF1
Host-side command: assert_reset
Host-side command: loadelf  C_Examples/hello/hello.RV64.bare.elf
Host-side command: deassert_reset
```

Repeat for
next ELF
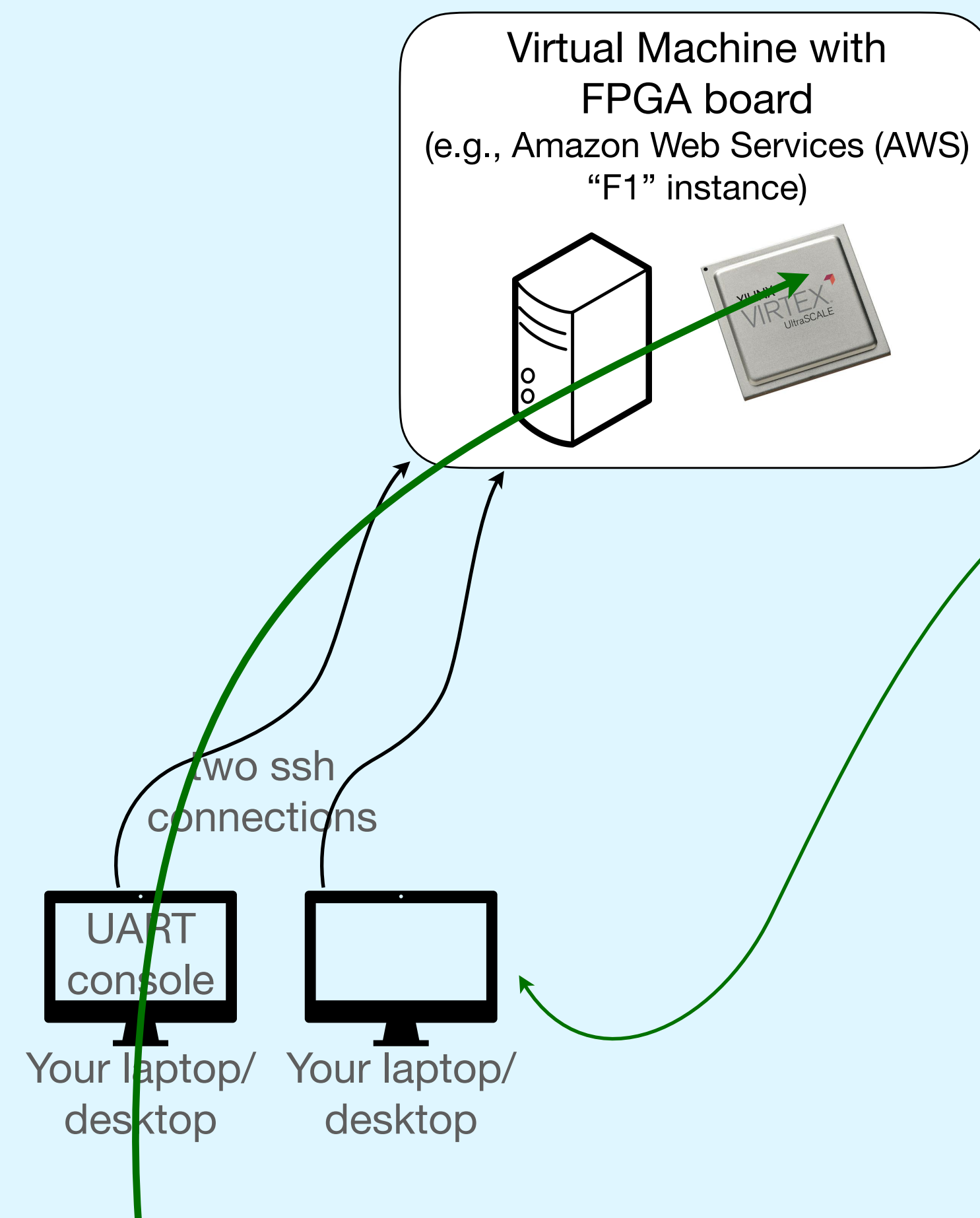
… observe "Hello World!" on UART console …

Demo/description of what you can do with Catamaran/ARIFIC

- DEMO: Cross-compile and run ISA tests

- DEMO: Cross-compile and run bare-metal (no OS) C programs

- *DEMO: Run Linux, with networking and block devices*

- DEMO: Cross-compile and run C programs under Linux

What you can do with Catamaran/ARIFIC
DEMO: Run Linux

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

two ssh
connections

UART
console

Your laptop/
desktop

Your laptop/
desktop

… interact with the Linux shell

Load bitfile into FPGA (Catamaran+Rocket):

```
$ sudo  fpga-load-local-image  -S 0  -I agfi-0e54cfdbc5e783a9b
```

Linux is just another ELF file, just like our bare-metal C programs

```
$ cd  Host_side_exes
$ ./exe_Host_AWSF1
Host-side command: assert_reset
Host-side command: loadelf     <path to Linux ELF>
Host-side command: deassert_reset
```

Observe console messages from OpenSBI boot loader,
followed by console messages from Linux boot,
finally the Linux shell prompt.
Then, …

*But note: without networking or block devices (disks), Linux is very limited*

What you can do with Catamaran/ARIFIC
DEMO: Run Linux with a block device (1/2)

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

two ssh
connections

UART
console

Your laptop/
desktop

Your laptop/
desktop

Create a file on the Ubuntu AMI host that will act as a block-device (disk) for Linux:

```
$ truncate — size=1G  disk.img
```

This represents a 1 GiB "unformatted" disk; we will
format it with a filesystem from inside Linux.
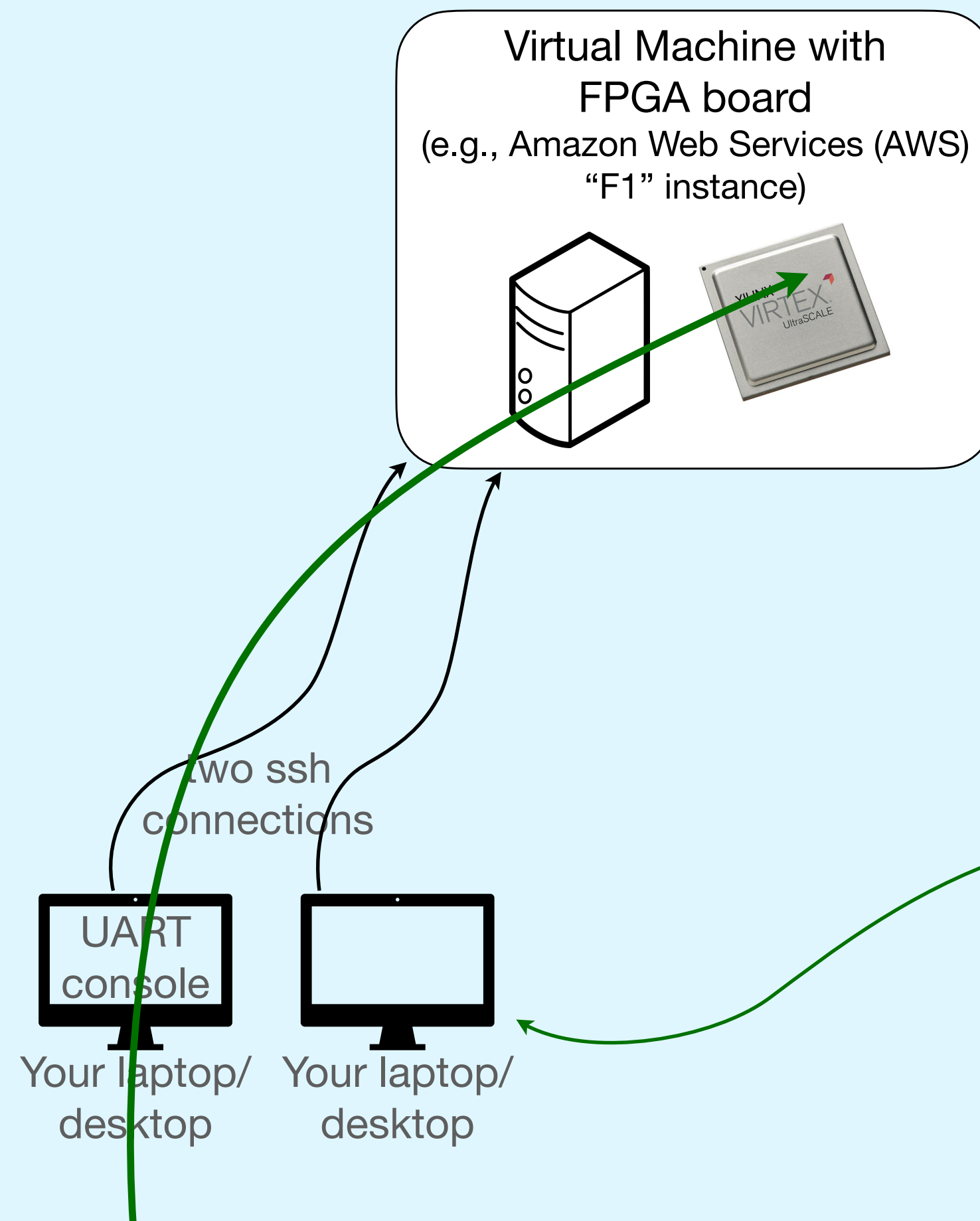
Boot Linux again on the FPGA, providing it these facilities:

```
$ sudo ./exe_Host_AWSF1. --elf ./Elfs/Linux.elf \
                         --blockdev  ./disk.img \
                         --tundev tap0
```

… interact with the Linux shell and use block device (see next slide)

What you can do with Catamaran/ARIFIC
DEMO: Run Linux with block device (2/2)

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

VIRTEX
UltraSCALE

two ssh
connections

UART
console

Your laptop/
desktop

Your laptop/
desktop

… interact with the Linux shell and use block device

After booting Linux, observe there is a block device:

```
# ls  /dev/vda
```
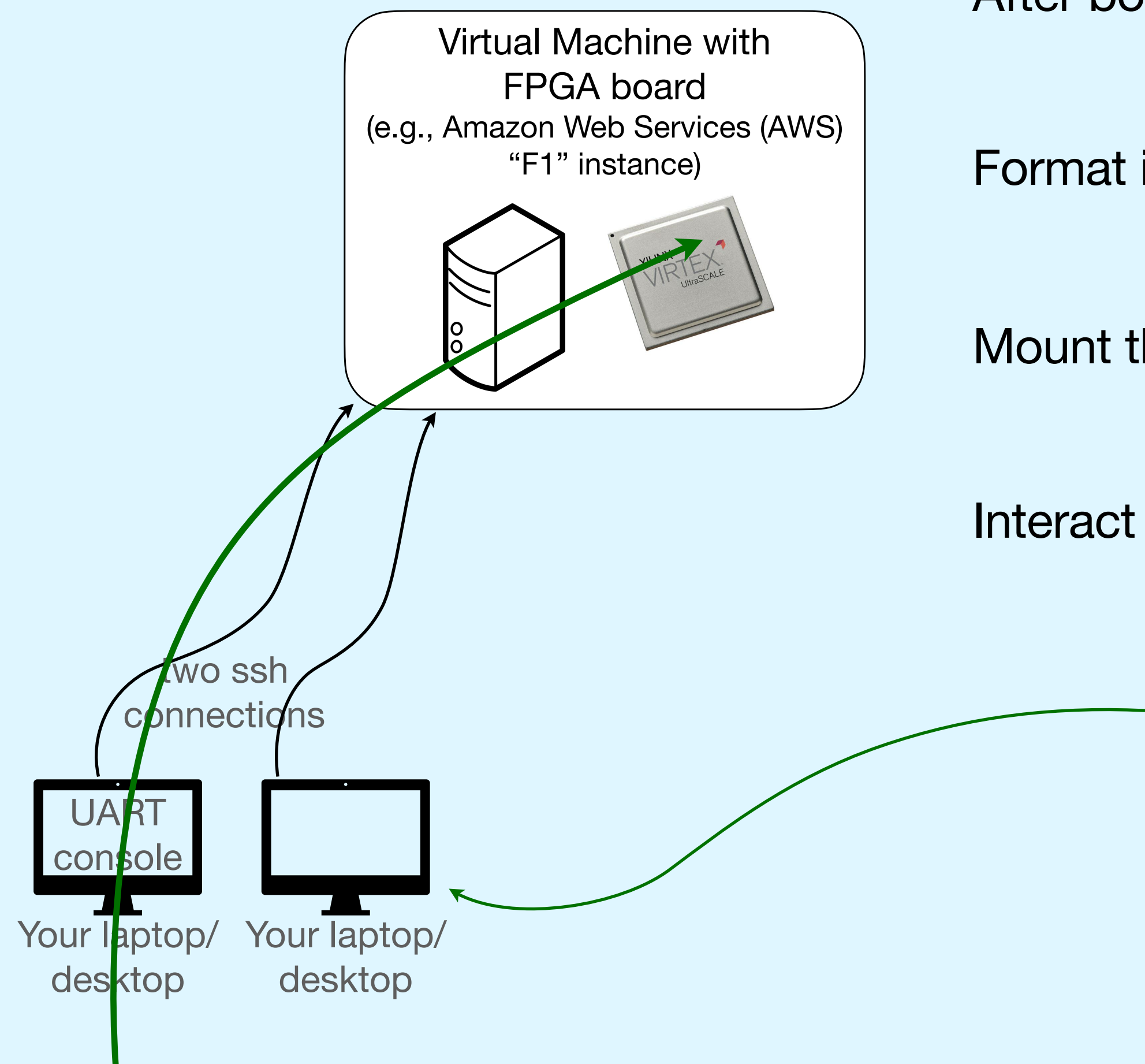
Format it with a filesystem:

```
# mkfs.ext2  /dev/vda
```

Mount the filesystem:
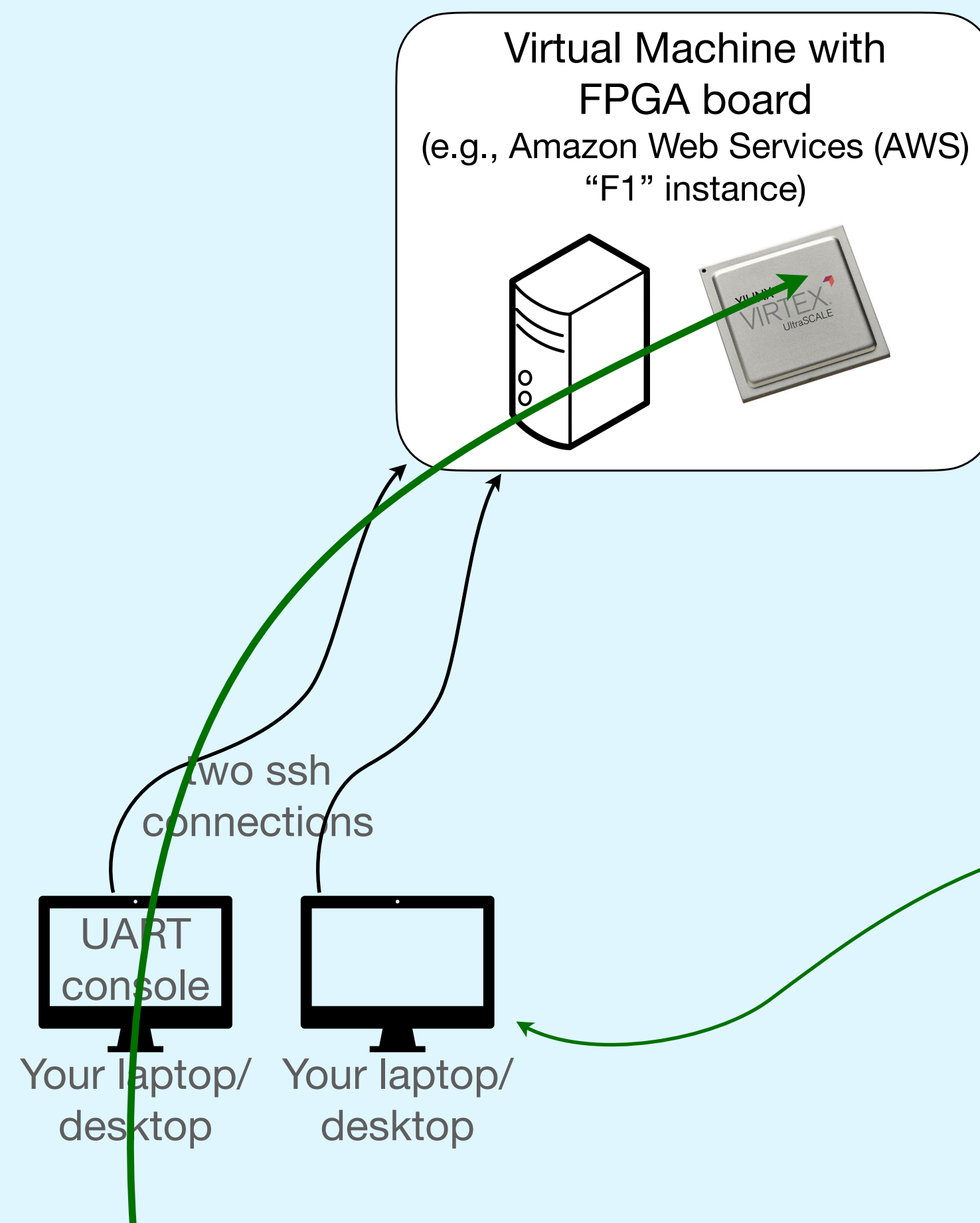
```
# mount  -t ext2  /dev/vda  /mnt
```

Interact with the filesystem

```
# ls ..
# cat …
```

What you can do with Catamaran/ARIFIC
DEMO: Run Linux with a network and block device (1/2)

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

two ssh
connections

UART
console

Your laptop/
desktop

Your laptop/
desktop

Execute the provided shell script "netinit.sh"

```
$ sudo  ./netinit.sh
```

This sets up a TUN/TAP networking device on your Ubuntu AMI,
through which Linux on the RISC-V will access the network.
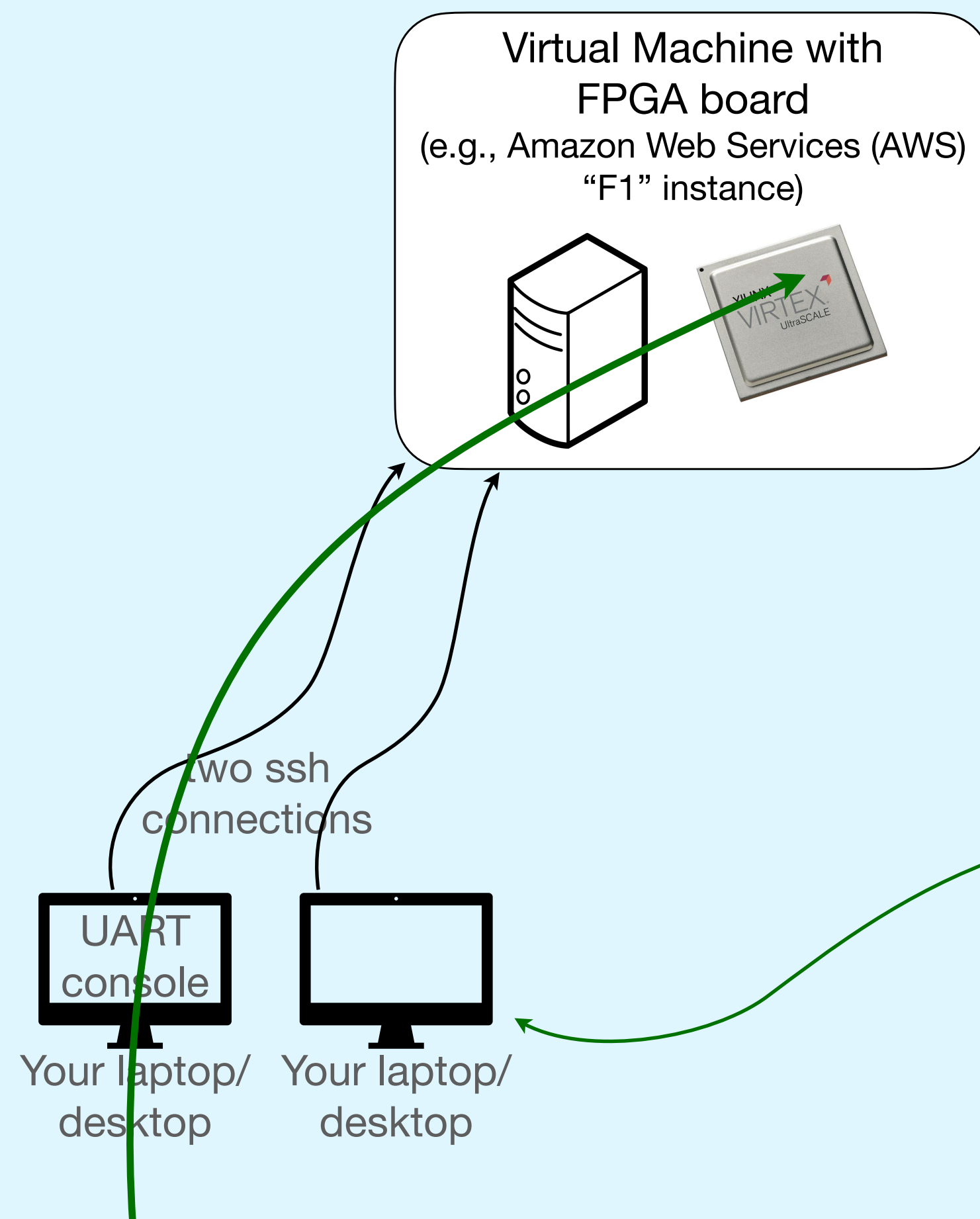
Boot Linux again on the FPGA, providing it these facilities:

```
$ sudo ./exe_Host_AWSF1. --elf ./Elfs/Linux.elf \
                         --blockdev  ./disk.img \
                         --tundev tap0
```

… interact with the Linux shell and use network and block device (see next slide)

What you can do with Catamaran/ARIFIC
DEMO: Run Linux with networking and block device (2/2)

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

two ssh
connections

UART
console

Your laptop/
desktop

Your laptop/
desktop

… interact with the Linux shell and use network and block device

After booting Linux, mount the block device:

```
# mount  -t ext2  /dev/vda  /mnt
```

Observe this network-setup shell script:

```
# cat  /mnt/net.sh
ip a add 192.168.3.2/24 dev eth0
ip link set eth0 up
ip route add 0.0.0.0/0 via 192.168.3.1
echo "nameserver 9.9.9.9" > /etc/resolv.conf
```

Execute it:

```
# /mnt/net.sh
```

Use 'ssh' to log in to a remote machine, 'scp' to copy files, etc.:

```
# ssh  user@remote.machine.org
# scp  localfile   user@remonte.machine.org:~/remotefile
# scp  user@remonte.machine.org:~/remotefile  localfile
```
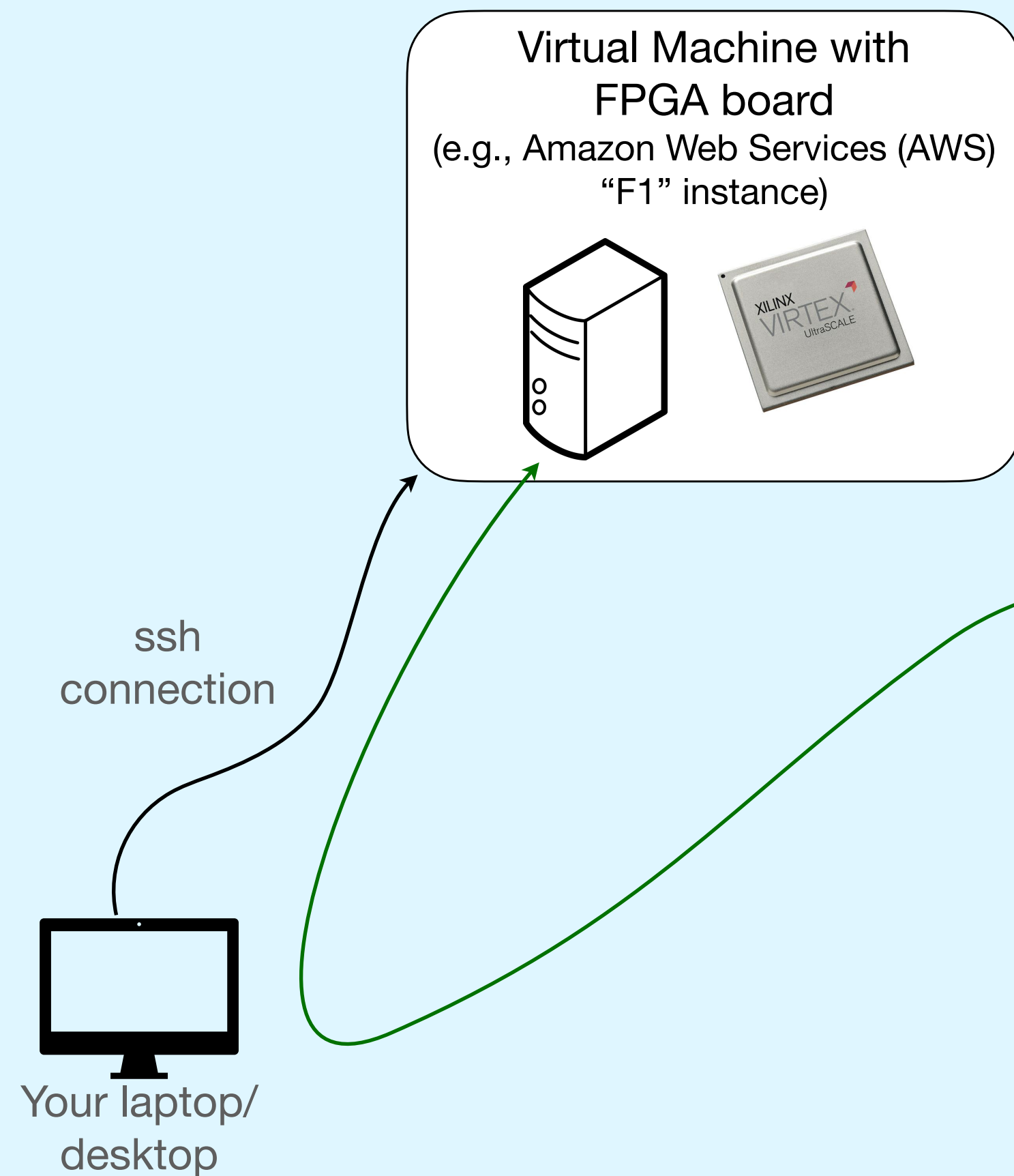
Demo/description of what you can do with Catamaran/ARIFIC

- DEMO: Cross-compile and run ISA tests

- DEMO: Cross-compile and run bare-metal (no OS) C programs

- DEMO: Run Linux, with networking and block devices

- *DEMO: Cross-compile and run C programs under Linux*

What you can do with Catamaran/ARIFIC
DEMO: Cross-compile and run C programs under Linux

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

ssh
connection

Your laptop/
desktop

Prerequisites:
- your RISC-V Gnu Toolchain (including cross-compiler gcc) is installed
- you've defined RISCV environment variable to point at your toolchain installation directory
- you've added ${RISCV}/bin to your PATH environment variable.

Classical "Hello World!" program

```
$ cd  C_Examples/hello
$ make  all_linux
```
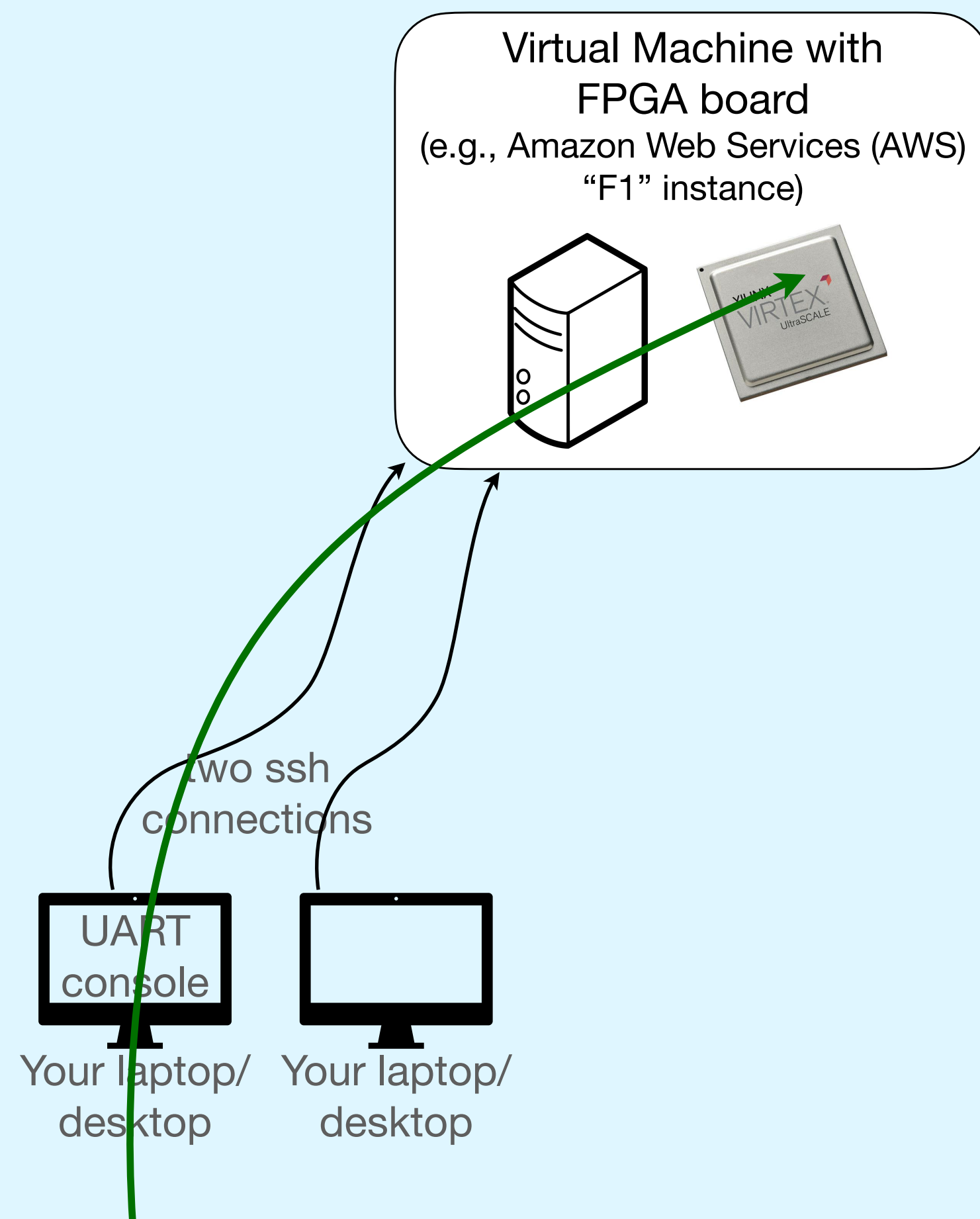
Creates: hello.RV64.linux.elf hello.RV64.linux.map hello.RV64.linux.objdump

Program that just echoes stdin to stdout (UART-in to UART-out)

```
$ cd  C_Examples/cat
$ make  all_linux
```

34

What you can do with Catamaran/ARIFIC
DEMO: Run C programs under Linux

Virtual Machine with
FPGA board
(e.g., Amazon Web Services (AWS)
"F1" instance)

VIRTEX
UltraSCALE

two ssh
connections

UART
console

Your laptop/
desktop

Your laptop/
desktop

At the Linux shell running on the RISC-V CPU, use 'scp' to copy the ELF files we created earlier for 'hello' and 'cat' for running under Linux:

```
# ls  /mnt/
… hello.RV64.linux.elf …
… cat.RV64.linux.elf …
```

Execute them:

```
# /mnt/hello.RV64.linux.elf
Hello, World!
# /mnt/cat.RV64.linux.elf
… echo chars typed on the keyboard to the screen …
```

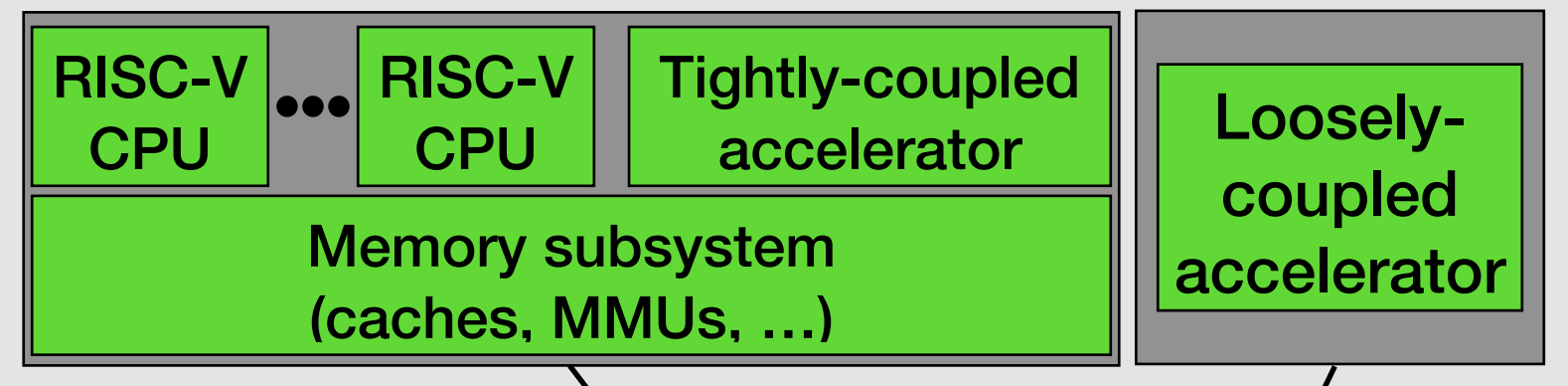… interact with the Linux shell and use network and block device

Demo/description of plugging in your own RISC-V Core

- The Catamaran/ARIFIC Core Interface (RTL)

- DEMO: Simulation flow (whole system, using Verilator)

- DEMO: FPGA flow on Amazon AWS

Plugging in your own RISC-V core: goal

Architecture Researcher's Focus
- CPU microarchitecture
- Memory systems (caches, MMUs, coherence, WMMs, …)
- Accelerators

RISC-V CPU	•••	RISC-V CPU	Tightly-coupled accelerator

Memory subsystem (caches, MMUs, …)

Loosely-coupled accelerator

Catamaran/ARIFIC (infrastructure in the cloud)

Host computer
running Linux

System Control:
- Load ELF/ memhex (programs, data)
- into DDR
- Assert/Deassert Core's RESET
- Configure CPU
- Collect stats, traces

Console (to/from UART)

Disk-device support

Network-device support

GDB control of CPU on FPGA

PCIe connection

FPGA

UART (for console)

Disk-device support

Network-device support

Support for counters, traces, etc.

(substitutable RISC-V core)

DDR memory

- Assume you've created your own core[1]

- Build a whole-system simulation executable of the FPGA-side, with your core substituted in place of the demo cores

- Build the FPGA bitfile of the FPGA-side, with your core substituted in place of the demo cores

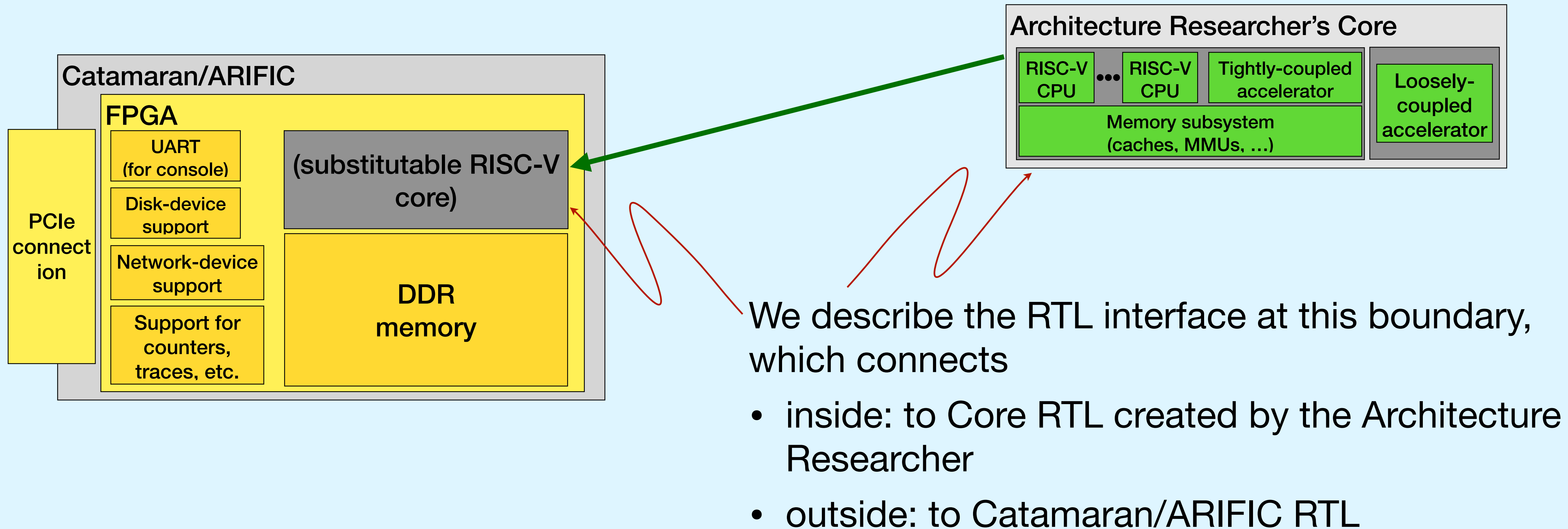[1] "Your own RISC-V Core"
Options:
- Your own new CPU design
- Modify available CPU (many open-source)
  - microarchitecture change
  - new instruction
  - new CSRs (e.g., counters)
- Modify/replace memory system (caches, MMUs, PMPs, PTWs, …)
- Add tightly-coupled or loosely coupled accelerator
- … or other research idea …

Demo/description of plugging in your own RISC-V Core

- *The Catamaran/ARIFIC Core Interface (RTL)*

- DEMO: Building a whole-system simulation executable

- DEMO: Building an FPGA bitfile for Amazon AWS

Plugging in your own RISC-V core
The Catamaran/ARIFIC Core Interface (RTL)

Architecture Researcher's Core

RISC-V CPU **...** RISC-V CPU Tightly-coupled accelerator

Loosely-coupled accelerator

Memory subsystem (caches, MMUs, …)

Catamaran/ARIFIC

FPGA

UART (for console)

Disk-device support

Network-device support

Support for counters, traces, etc.

(substitutable RISC-V core)

DDR memory

PCIe connection

We describe the RTL interface at this boundary, which connects

- inside: to Core RTL created by the Architecture Researcher
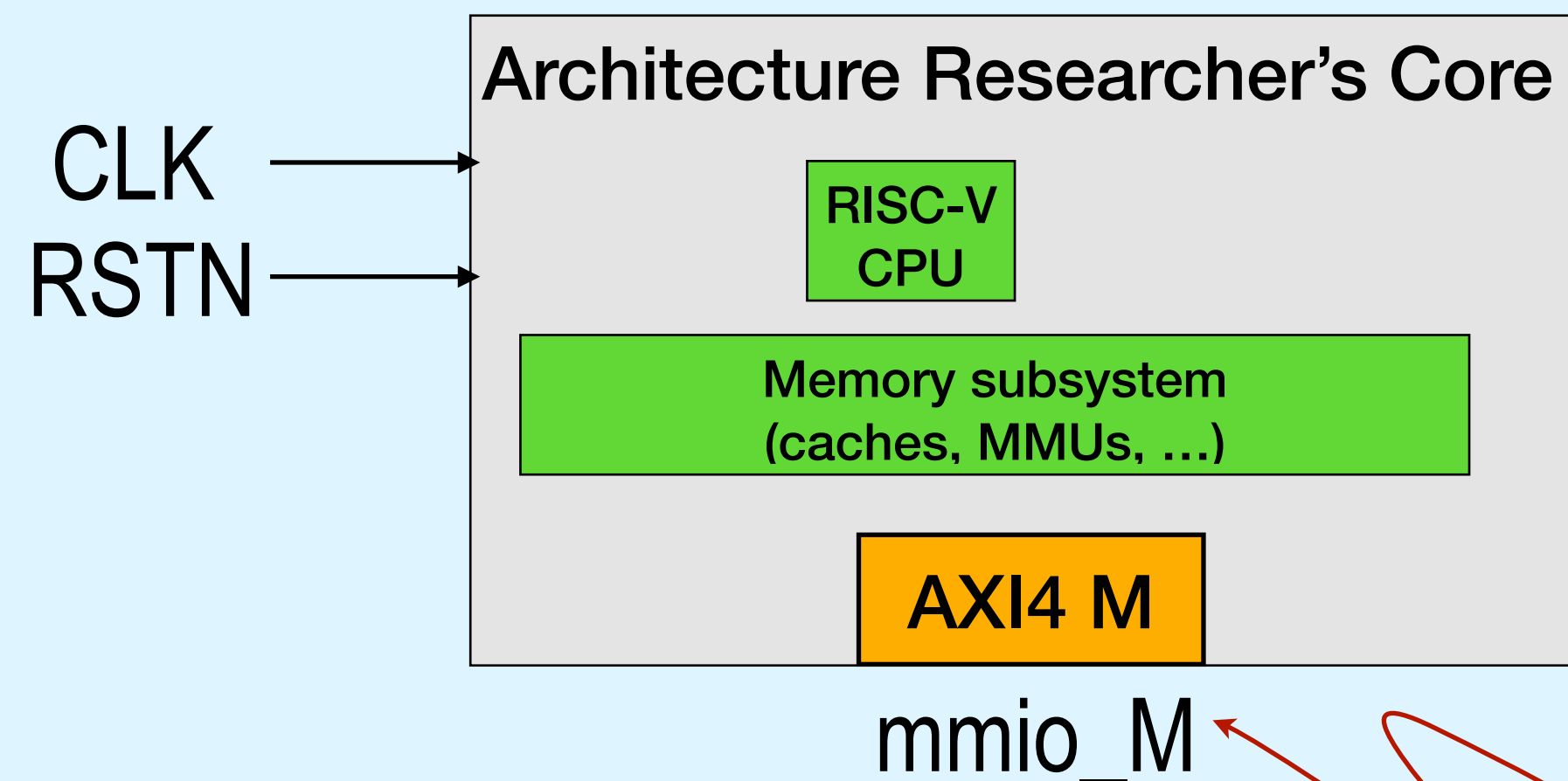- outside: to Catamaran/ARIFIC RTL

Plugging in your own RISC-V core
The Catamaran/ARIFIC Core Interface (RTL)

Demo: View the actual interface RTL in the tutorial repository

Minimal core

CLK
RSTN

Architecture Researcher's Core

RISC-V
CPU

Memory subsystem
(caches, MMUs, …)

AXI4 M

mmio_M

CPU can be RV32 or RV64,

from small ("embedded, IoT")

to large ("application", "server").

AXI4 "manager" interface
64-bit address, 64-bit data,
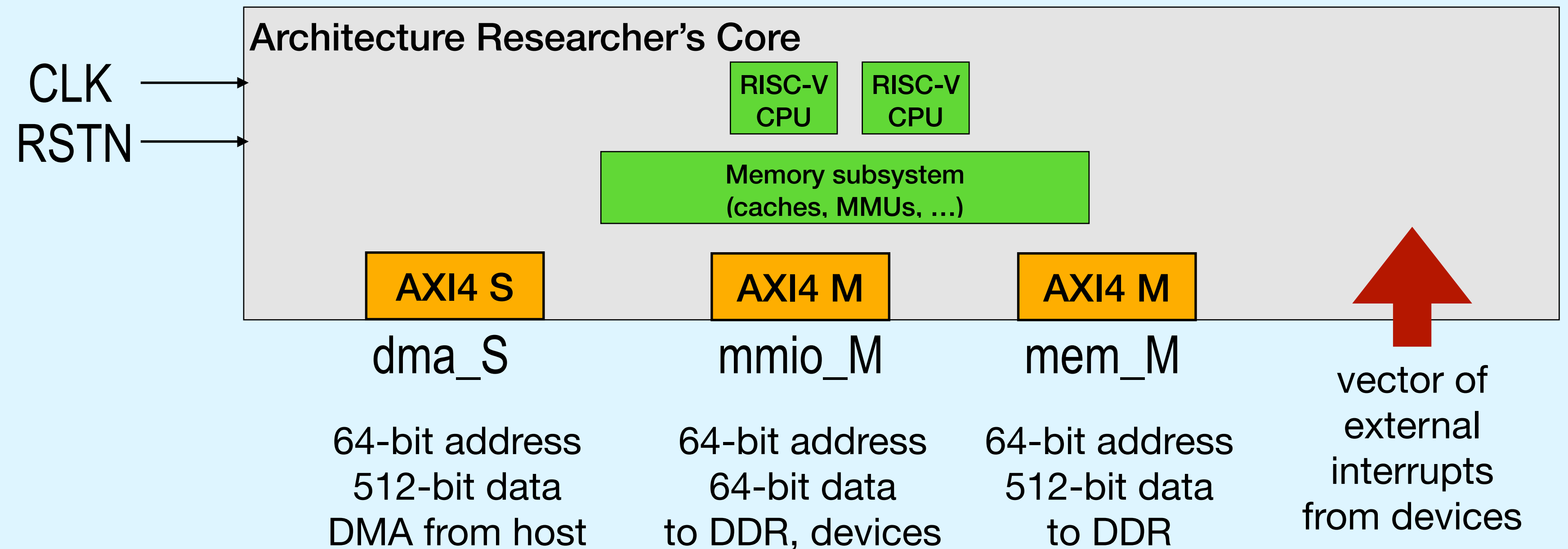connects to DDR memory,
UART, other devices

CLK can connect to one of six available clocks provided by Catamaran/ARIFIC, ranging from about 15 MHz to 250 MHz, depending on the speed/timing requirements of the core.

Plugging in your own RISC-V core
The Catamaran/ARIFIC Core Interface (RTL)

Demo: View the actual interface RTL in the tutorial repository

CLK
RSTN

More interfaces

Architecture Researcher's Core

RISC-V CPU   RISC-V CPU

Memory subsystem
(caches, MMUs, …)

AXI4 S   AXI4 M   AXI4 M

dma_S   mmio_M   mem_M

vector of external interrupts from devices

64-bit address
512-bit data
DMA from host

64-bit address
64-bit data
to DDR, devices

64-bit address
512-bit data
to DDR
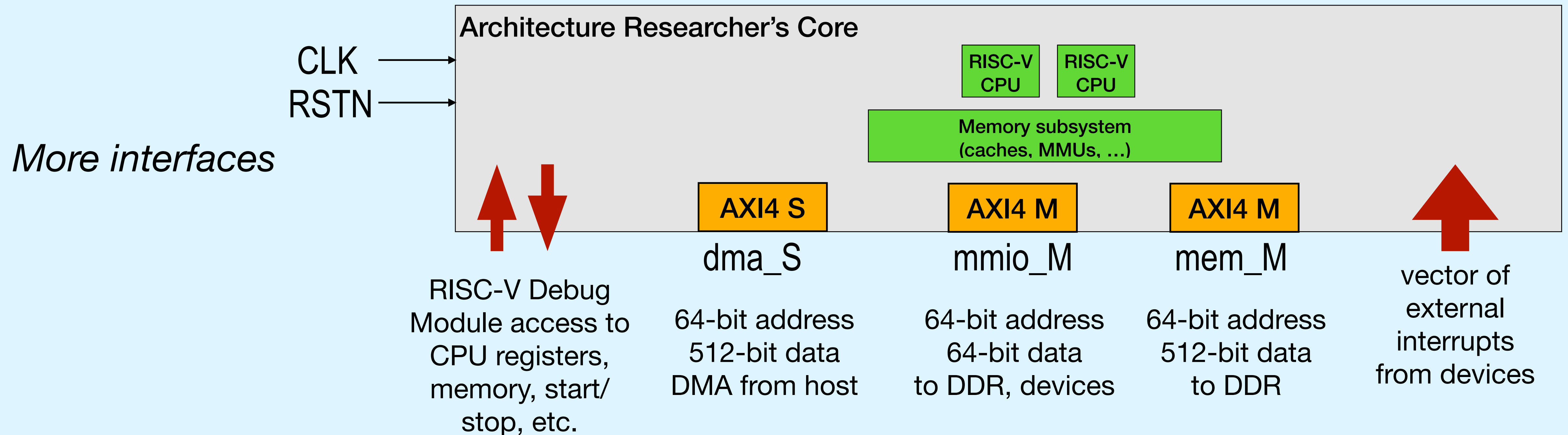
- In some cores, mem_M transfers entire cache lines at the back-end of an L2 cache
- In some cores, dma_S connects to a cache-coherent port in the memory subsystem

41

Plugging in your own RISC-V core
The Catamaran/ARIFIC Core Interface (RTL)

Demo: View the actual interface RTL in the tutorial repository

CLK

RSTN

More interfaces

Architecture Researcher's Core

RISC-V CPU   RISC-V CPU

Memory subsystem
(caches, MMUs, …)

AXI4 S

dma_S

RISC-V Debug
Module access to
CPU registers,
memory, start/
stop, etc.

64-bit address
512-bit data
DMA from host

AXI4 M

mmio_M

64-bit address
64-bit data
to DDR, devices

AXI4 M

mem_M

64-bit address
512-bit data
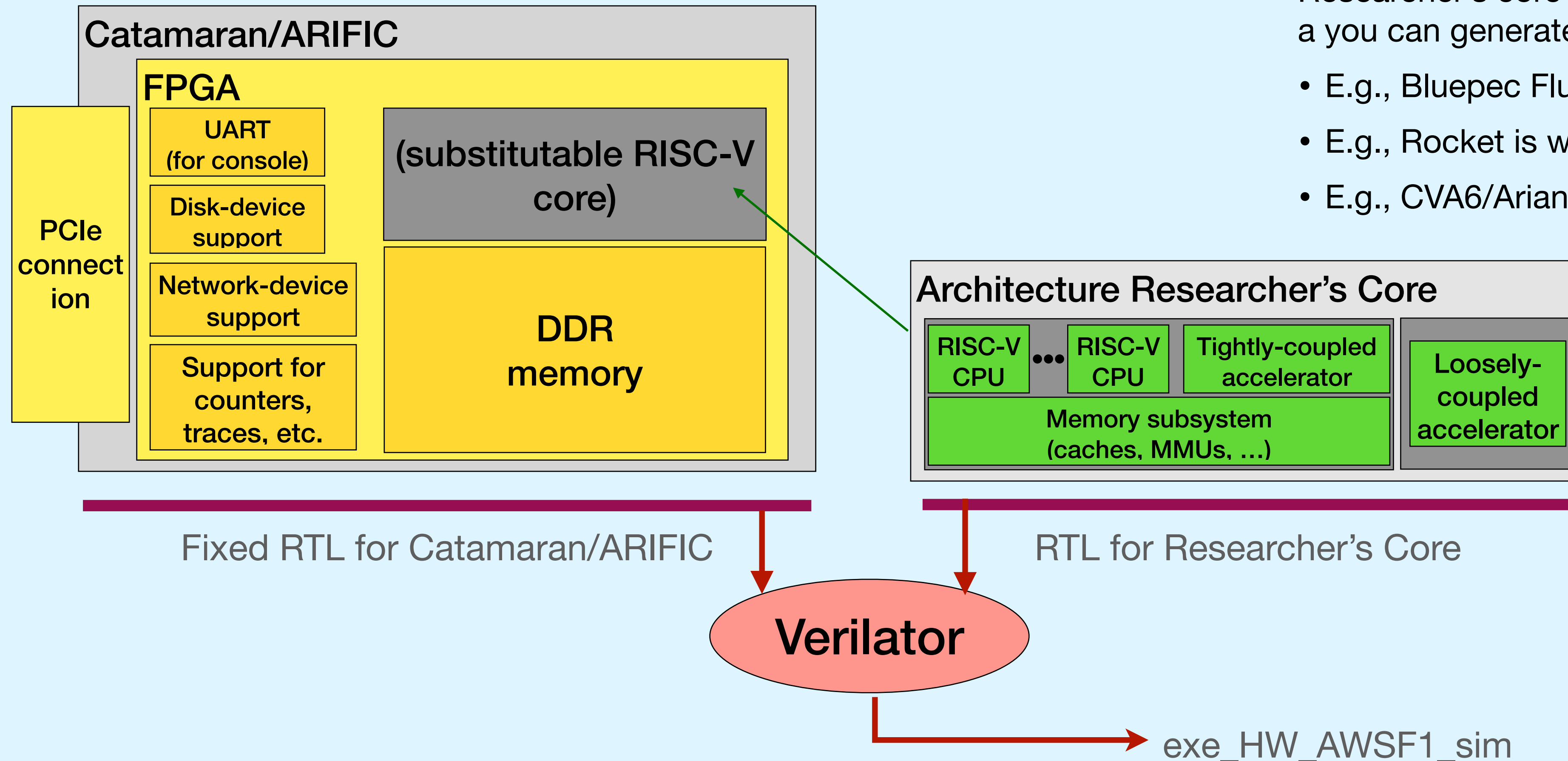to DDR

vector of
external
interrupts
from devices

Demo/description of plugging in your own RISC-V Core

- The Catamaran/ARIFIC Core Interface (RTL)

- *DEMO: Building a whole-system simulation executable*

- DEMO: Building an FPGA bitfile for Amazon AWS

Plugging in your own RISC-V core
Building a whole-system simulation executable

Catamaran/ARIFIC

FPGA

PCIe connection

UART (for console)

Disk-device support

Network-device support

Support for counters, traces, etc.

(substitutable RISC-V core)

DDR memory

Fixed RTL for Catamaran/ARIFIC

Architecture Researcher's Core

RISC-V CPU ••• RISC-V CPU

Tightly-coupled accelerator

Loosely-coupled accelerator

Memory subsystem (caches, MMUs, …)

RTL for Researcher's Core

Verilator

exe_HW_AWSF1_sim

Researcher's core can be written in any HDL, as long a you can generate RTL for it.

- E.g., Bluepec Flute is written in BSV
- E.g., Rocket is written in Chisel
- E.g., CVA6/Ariane is written in SystemVerilog

Plugging in your own RISC-V core
Demo: Building a whole-system simulation executable

This tutorial repository contains

- Fixed RTL for Catamaran/ARIFIC

- RTL for Bluespec Flute RISC-V Core (RV64GC_MSU_Sv39)

- Makefile for using Verilator to create a whole-system simulation executable

  - Note: Verilator is a free, open-source tool for creating RTL simulators

  - For installation, please see: https://verilator.org/guide/latest/install.html
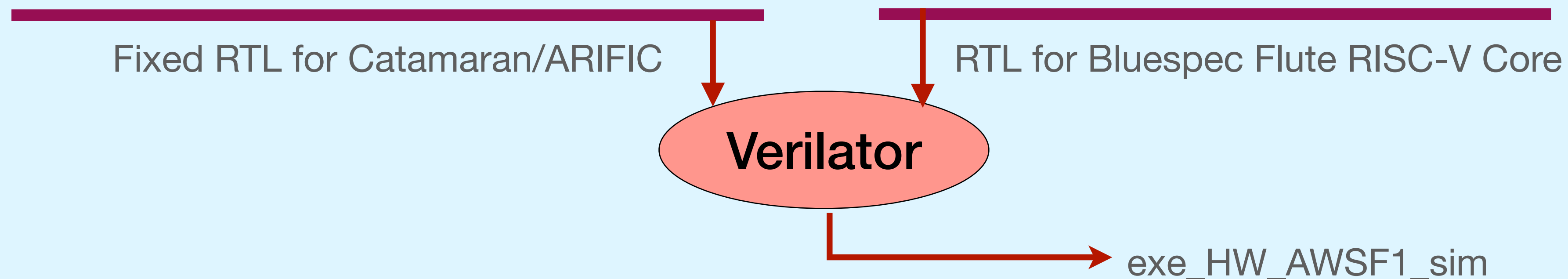
Fixed RTL for Catamaran/ARIFIC

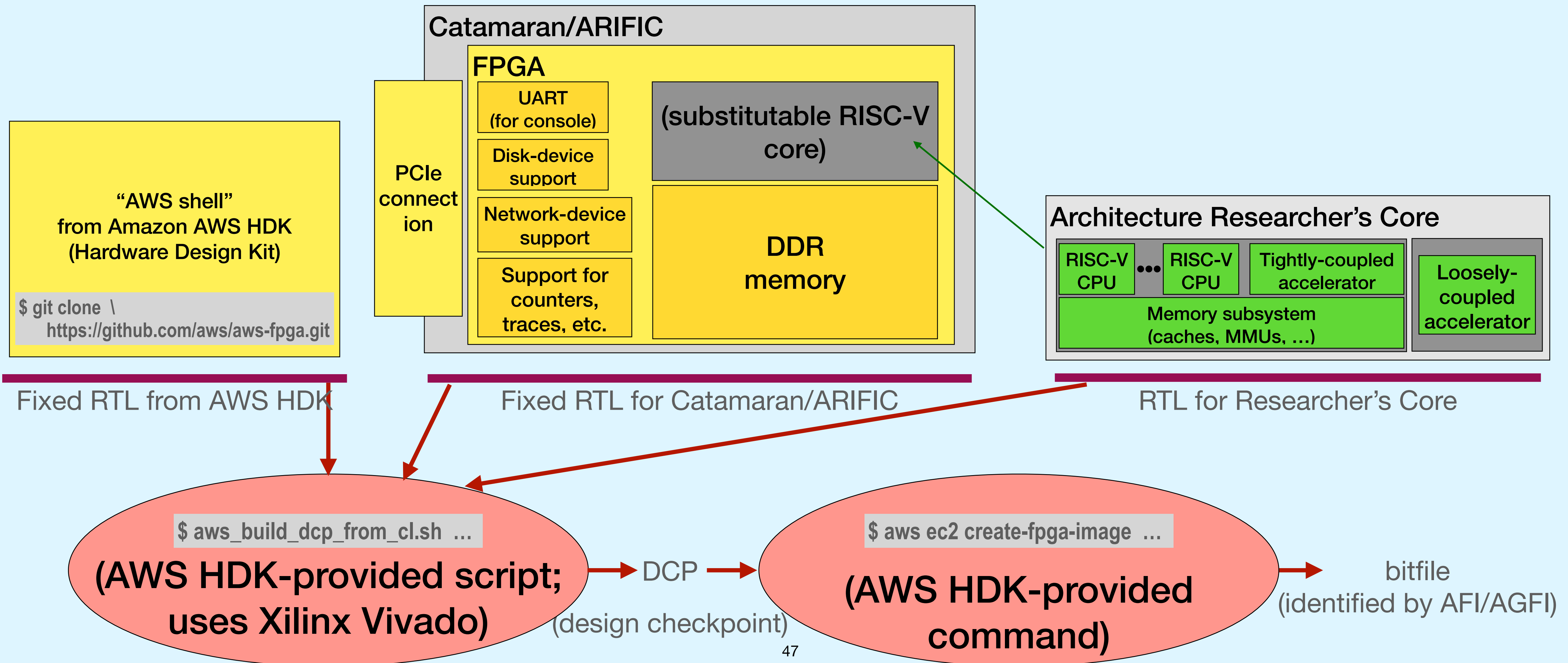RTL for Bluespec Flute RISC-V Core

Verilator

exe_HW_AWSF1_sim

Demo/description of plugging in your own RISC-V Core

- The Catamaran/ARIFIC Core Interface (RTL)

- DEMO: Simulation flow (whole system, using Verilator)

- *DEMO: Building an FPGA bitfile for Amazon AWS*

Plugging in your own RISC-V core
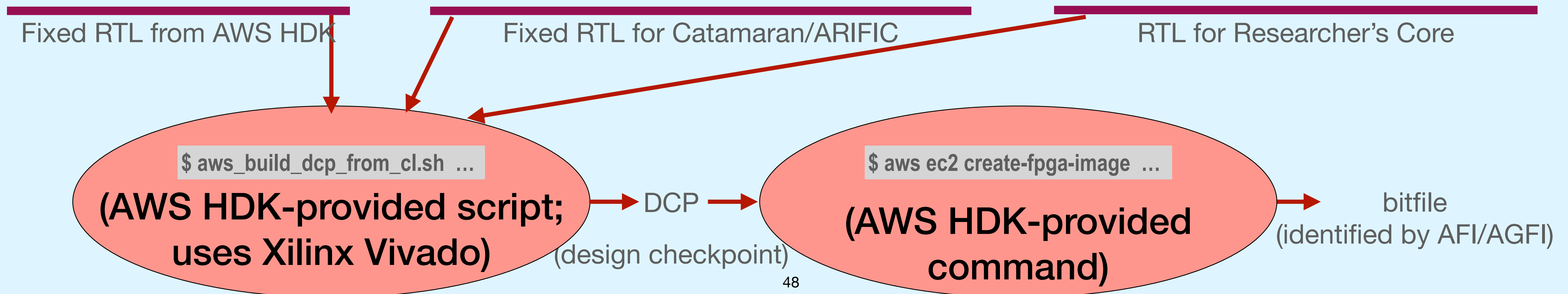## Building an FPGA bitfile for Amazon AWS

Catamaran/ARIFIC

FPGA

UART
(for console)

(substitutable RISC-V core)

Disk-device support

Network-device support

PCIe connection

DDR memory

Support for counters, traces, etc.

"AWS shell"
from Amazon AWS HDK
(Hardware Design Kit)

$ git clone \
  https://github.com/aws/aws-fpga.git

Architecture Researcher's Core

RISC-V CPU ••• RISC-V CPU | Tightly-coupled accelerator | Loosely-coupled accelerator

Memory subsystem
(caches, MMUs, ...)

Fixed RTL from AWS HDK

Fixed RTL for Catamaran/ARIFIC

RTL for Researcher's Core

$ aws_build_dcp_from_cl.sh ...

(AWS HDK-provided script; uses Xilinx Vivado)

DCP

(design checkpoint)

$ aws ec2 create-fpga-image ...

(AWS HDK-provided command)

bitfile
(identified by AFI/AGFI)

Plugging in your own RISC-V core
Demo: Building an FPGA bitfile for Amazon AWS

During the tutorial, we shall only demo the commands initiating these two steps, and not wait for the results
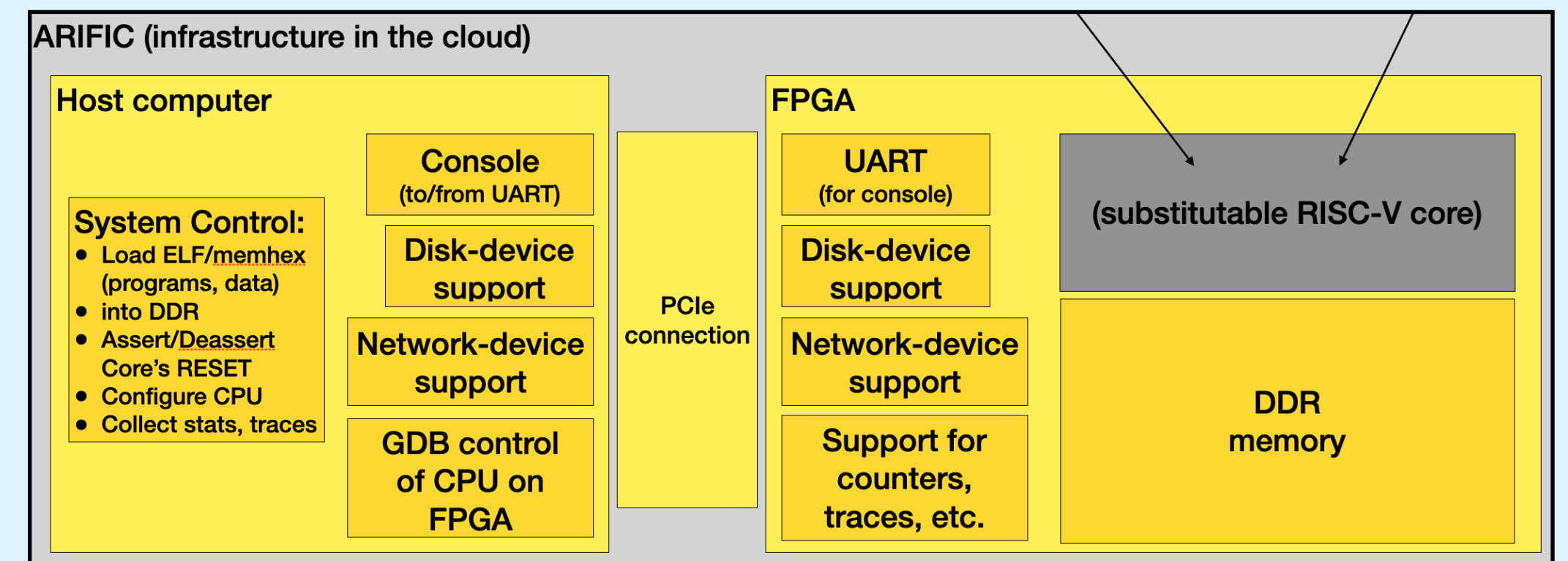- The "build DCP" step can take several hours (e.g., 2h30m for building with CVA6 CPU)
- The "create FPGA image" step takes about 1 hour

Fixed RTL from AWS HDK   Fixed RTL for Catamaran/ARIFIC   RTL for Researcher's Core

`$ aws_build_dcp_from_cl.sh …`

(AWS HDK-provided script; uses Xilinx Vivado)

DCP

(design checkpoint)

`$ aws ec2 create-fpga-image …`

(AWS HDK-provided command)

bitfile
(identified by AFI/AGFI)

48

Use GDB to control the RISC-V CPU on the FPGA

Summary and Conclusion

Summary of tutorial

ARIFIC (infrastructure in the cloud)

Host computer

System Control:
- Load ELF/memhex (programs, data)
- into DDR
- Assert/Deassert Core's RESET
- Configure CPU
- Collect stats, traces

Console (to/from UART)

Disk-device support

Network-device support

GDB control of CPU on FPGA

PCIe connection

FPGA

UART (for console)

Disk-device support

Network-device support

Support for counters, traces, etc.

(substitutable RISC-V core)

DDR memory

- Catamaran/ARIFIC provides the large and complex infrastructure that frees the Architecture Researcher to focus on their research into the CPU microarchitectures, memory systems, accelerators, etc.

- In this tutorial we showed the capabilities thereby immediately enabled for the Researcher:

  - Run on FPGA on Amazon AWS virtual machines

  - Run ISA tests

  - Run bare-metal C programs with console I/O

  - Run Linux and C programs-under-Linux with console I/O, networking and "disk" devices

  - GDB-debug bare-metal programs

  - Dump memory data (e.g., performance data collected during program execution)

End of Tutorial Slides

Appendix with reference material follows

Thank you all for attending!

GitHub repo for this tutorial: https://github.com/rsnikhil/Tutorial_at_HPCA-29

Appendix: Reference Material

- *Installing the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)*

- Opening an account on Amazon AWS

- Creating an AMI (or two) for Catamaran/ARIFIC

- Installing aws-fpga HDK and SDK

- Installing XDMA driver for Host-FPGA PCIe communication

Installing the RISC-V Gnu Toolchain

- By "RISC-V Gnu Toolchain" we mean the familiar *gcc, as, ld, gdb, objdump*, … etc.

- These are needed for compiling C, C++ and RISC-V Assembly Language programs into ELF binaries to be loaded and run on a RISC-V CPU.

- We install "cross-compilers" that run under Ubuntu, but generate RISC-V machine code.

Full details may be found at this URL; please follow the installation directions there.

https://github.com/riscv/riscv-gnu-toolchain

Hint: in the "configure" step, specify "medany" and "multilib":

```
$ ./configure  --prefix=<path_to_installation_dir> --with-cmodel=medany --enable-multilib
```

- <path…> should be a full absolute path (do not use ~ for your home directory).
- "newlib" is for the compiler for bare-metal C RV32 and RV64 programs (no system calls)
- "multilib" produces a compiler that can produce both RV32 and RV64 code, even though it is named *riscv64-unknown-elf-gcc*

Before using the cross-compiler, export the following environment variables:

```
$ export RISCV=<path_to_installation_dir>
$ export PATH=${RISCV}/bin:${PATH}
```

Appendix: Reference Material

- Installing the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)

- *Opening an account on Amazon AWS*

- Creating an AMI (or two) for Catamaran/ARIFIC

- Installing aws-fpga HDK and SDK

- Installing XDMA driver for Host-FPGA PCIe communication
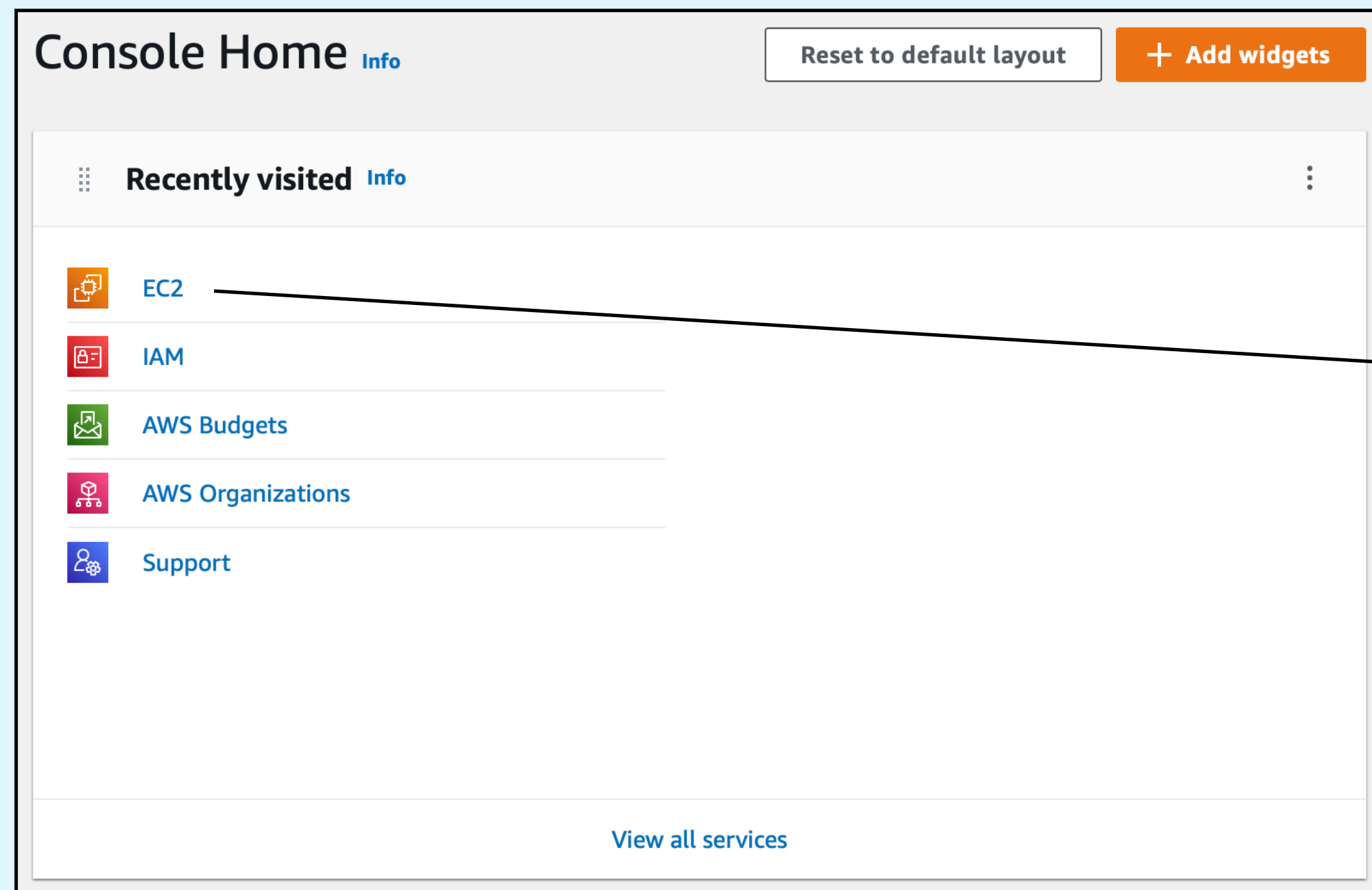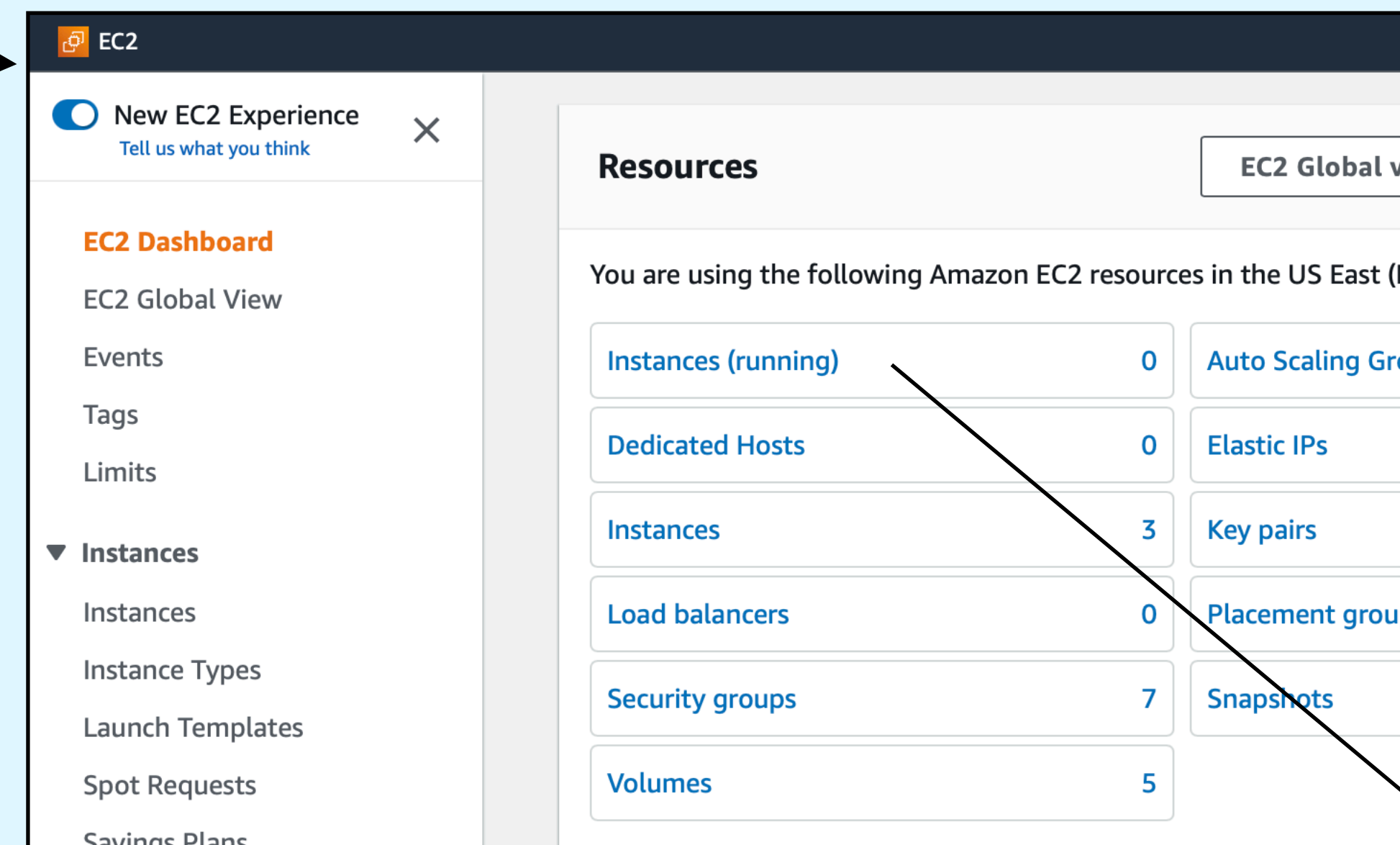
Create your Amazon AWS account

- At: https://aws.amazon.com
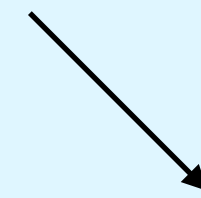
First time

Subsequently

Amazon runs cloud farms in many parts of the world; you may be asked to select a region "near" you.
E.g., I use: 'us-east-1 (N.Virginia)'
Caution: select a region that has support instances with FPGAs

56

Appendix: Reference Material

- Installing the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)

- Opening an account on Amazon AWS

- *Creating an AMI (or two) for Catamaran/ARIFIC*

- Installing aws-fpga HDK and SDK

- Installing XDMA driver for Host-FPGA PCIe communication

Navigate to your EC2 Instances dashboard

Console Home Info

| Reset to default layout | + Add widgets |

⠿ **Recently visited** Info   ⋮

🔲 EC2

🔳 IAM

📇 AWS Budgets

🔲 AWS Organizations

🔲 Support

View all services

AWS offers a gazillion services; we want "EC2" (Elastic Computing), which are virtual machines in the cloud.

🔲 EC2

🔘 New EC2 Experience   ✕
Tell us what you think

EC2 Dashboard

EC2 Global View

Events

Tags

Limits

▼ **Instances**

Instances

Instance Types

Launch Templates

Spot Requests

Savings Plans

Resources   EC2 Global vi

You are using the following Amazon EC2 resources in the US East (N

Instances (running)	0	Auto Scaling Gro
Dedicated Hosts	0	Elastic IPs
Instances	3	Key pairs
Load balancers	0	Placement group
Security groups	7	Snapshots
Volumes	5	

● ● ●

Then, move on to the "Instances" dashboard ("instances" = AWS terminology for your virtual machines)

The EC2 "instances" dashboard

Create new instances from here:

Your existing instances are listed here. These persist across your AWS logins, until you delete them.

Select one or more instances here

These instances are all currently "Stopped" (like "sleep" on a laptop; saves AWS charges when not being actively used)
- Selected instances can be put into the "Running" state by selecting "Start instance" in the "Instance state" drop-down menu at top (like emerging from "sleep" on a laptop).

The "Instance type" determines:
- Host CPU type (x86, ARM, …)
- Sizing (# of virtual cores, memory, …)
- "f1.*" types are the only ones with FPGAs attached.
- *Note*: AWS charges vary with instance type
- *Warning*: "f1" instances typically have higher AWS charges.
- *Hint*: you can do FPGA builds on an instance without an FPGA (cheaper), such as the first one shown above.

Create one or two AMIs?

The "Instance type" determines, for your AMI:
- Host CPU type (x86, ARM, …)
- Sizing (# of virtual cores, memory, …)
- "f1.*" types are the *only* ones with FPGAs attached.
- *Note*: AWS charges vary with instance type
- *Warning*: "f1" instances typically have higher AWS charges.

Hint (for lower AMS charges): Create two AMIs:

- A "build" AMI of type "m6i.2xlarge" for building bitfiles for Catamaran/ARIFIC

  - Use the free AWS "FPGA Developer AMI" for this, which comes preloaded with Vivado, Vivado licenses, etc. (free). The supplied OS is CentOS Linux.

- A "run" AMI of type "f1.2xlarge" for running Catamaran/ARIFIC with FPGA

  - You can choose an AMI with OS of your choice (we typically choose latest Ubuntu)

Note: when you build a bitfile on the "build" AMI, it'll reside in the cloud named by an AGFI (unique id), which you can access from the "run" AMI

"Create" your virtual machine ("instance")

When you select "Launch Instances" to create a new instance, it will take you through a series of steps ("Launch Instances Wizard").
Here are some notes on the various steps:

- "Name and tags": choose a name for your instance (e.g., on our slides: "RSN_AFI_Build2", "RSN_AFI_run", "RSN_AWSteria")
- Application and OS Images (Amazon Machine Image)
    - Choose one of the AMIs from the 1000s available in the search box
    - For Catamaran/ARIFIC bitfile building, we recommend the latest AWS "FPGA Developers AMI"
    - For Catamaran/ARIFIC running on FPGA, we recommend an AMI with the latest Ubuntu (22.04 LTS at time of writing), x86 (not ARM!), 64-bit
- "Select and Instance Type"
    - The menu shows 100s of possibilities, x86 and ARM, etc.
    - For Catamaran/ARIFIC bitfile building, we recommend "m6i.2xlarge" (e.g., see "Instance type" on previous slide)
    - For Catamaran/ARIFIC running on FPGA, this *must* be an "f1" instance; we recommend "f1.2xlarge" (e.g., see "Instance type" on our slides)
- "Create your Key Pair"
    - When one connects to an AMI (with "ssh"), for security reasons Amazon does not support password-based login authentication, only crypto-key based authentication.
    - This menu Item creates a key pair for you, a public key and private key.
    - The prompts will ask you to copy the private key, and save it on your laptop/desktop in a file `~/.ssh/MyPrivateKey.pem` and set its protection to 0x400
    - Note: once you've created a key pair, you can reuse it for new instances; it will show you your existing key pairs and you can select one, instead of creating a new one.
- "Network Settings": Leave unchanged.   In particular, "Auto-assign public IP" = "Enable" and "Allow SSH traffic from Anywhere"
- "Configure Storage": Leave unchanged, e.g., 8 GiB (these can be updated later, if needed)
- "Advanced Details": Leave unchanged
- Finally, select "Launch Instance" on the "Summary" panel which floats on the right during the wizard.
    - (but see caveat on next slide)

"Create" your virtual machine ("instance")

Caveat: After selecting "Launch Instance" at the end of the AMI-creation wizard, you may encounter an error message like this:

> You have requested more vCPU capacity than your current vCPU limit of 0 allows for the instance bucket that the specified instance type belongs to. Please visit
>   http://aws.amazon.com/contact-us/ec2-request
> to request an adjustment to this limit.

If you get this message, please visit the contact-us link shown, click on "Support" and lodge a request to increase your "vCPU quota" to 8, which is required for the f1.2xlarge instance type. Try to include a few sentences explaining your need for an f1.2xlarge instance type.
You will get an almost instance response saying:
    This specific limit increase request requires further internal review before approval …
These requests are processed by humans, so it may take a few hours before you receive an email indicating that your request has been approved, after which you can proceed.

Appendix: Reference Material

- Installing the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)

- Opening an account on Amazon AWS

- Creating an AMI (or two) for Catamaran/ARIFIC

- *Installing aws-fpga HDK and SDK*

- Installing XDMA driver for Host-FPGA PCIe communication

Installing aws-fpga HDK and SDK

"aws-fpga" is a free HDK (Hardware Development Kit) + SDK (Software Development Kit)

- The HDK is needed for creating AWS bitfiles for the FPGA

- The SDK is needed for creating host-side software that communicates with the FPGA

- It also contains the Xilinx XDMA Linux driver for the host-side to communicate with the FPGA over the PCIe connection

Git-clone it into your AMI from here:

```
$ git clone  https://github.com/aws/aws-fpga.git
```

Each time you connect to your AMI, please do:

```
$ cd  aws-fpga
$ source  hdk_setup.sh
$ source  sdk_setup.sh
```

Appendix: Reference Material

- Installing the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)

- Opening an account on Amazon AWS

- Creating an AMI (or two) for Catamaran/ARIFIC

- Installing aws-fpga HDK and SDK

- *Installing XDMA driver for Host-FPGA PCIe communication*

Installing XDMA driver for Host-FPGA PCIe communication

The XDMA driver is a Linux kernel driver for the host to communicate with the FPGA over the high-speed PCIe bus.   The driver is included in the "aws-fpga" HDK + SDK.

Git-clone the aws-fpga HDK+SDK into your AMI from here:

```
$ git clone  https://github.com/aws/aws-fpga.git
```

Compile the driver (creates file "xdmi.ko"):

```
$ cd aws-fpga/sdk/linux_kernel_drivers/xdma
$ make
...
$ ls xdma.ko
```

Install the driver into the AMI's running Linux kernel:

```
$ sudo make install
```

Check that XDMA is installed and running:

```
$ lsmod | grep xdma
xdma                94208  0

$ ls /dev/xdma*
   ... will show many devices with the xdma prefix ...
```

END

GitHub repo for this tutorial: https://github.com/rsnikhil/Tutorial_at_HPCA-29