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?

The Architecture Researcher’s Dilemma

Creative work; occupies you full-time

- The Architecture Researcher is typically focused on a specific artifact:
  - CPU pipeline organization, branch predictor, cache organization, store buffers, instruction fusion, …, reorder buffer, accelerator…

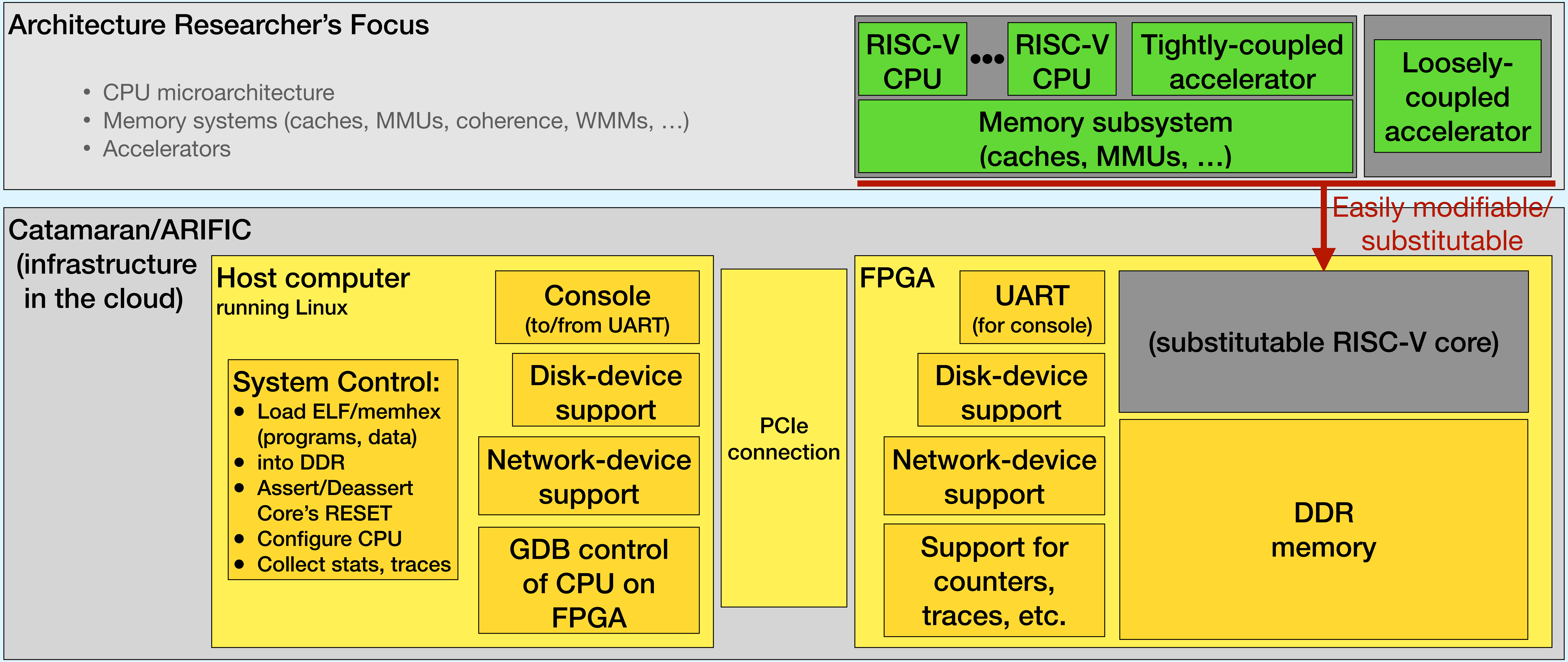
Lots of boring boilerplate; man-years to construct; not portable, reusable or scalable (e.g., one-off, soon-obsolete FPGA board, unfixable on departure of key grad student), …

- But for credible impact, needs whole-system environment:
  - Reasonable SoC with devices (networking, block storage, UART)
  - Running actual apps, often under a real operating system
  - On FPGA
    - For microarchitecture credibility
    - To run programs of credible length

Beyond Qemu, GEM5, …

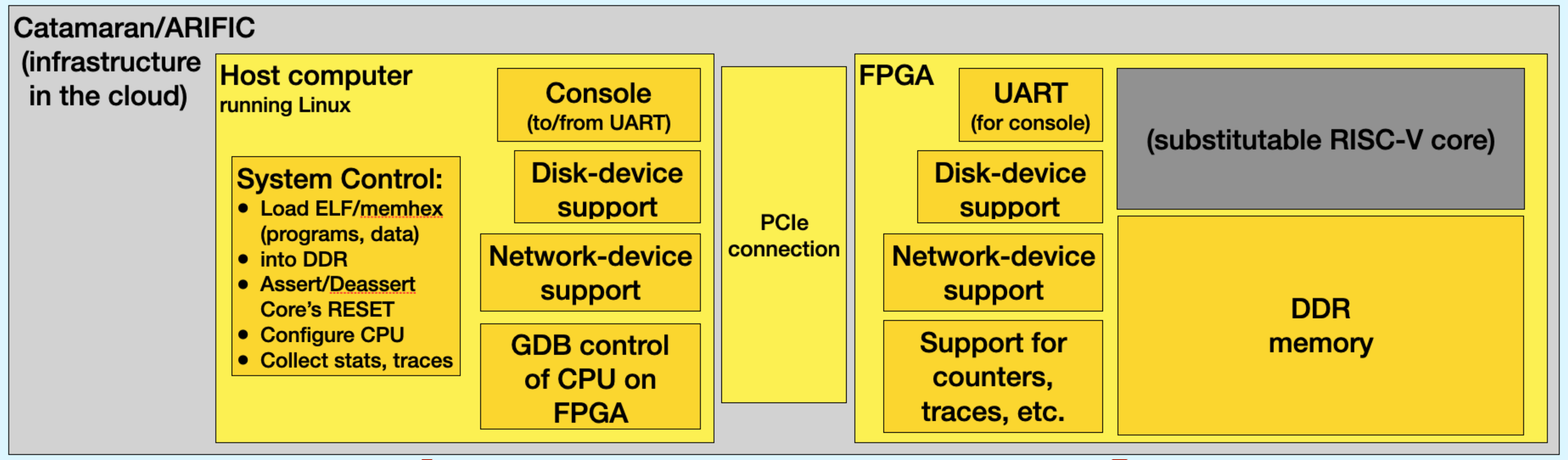
What is Catamaran/ARIFIC?

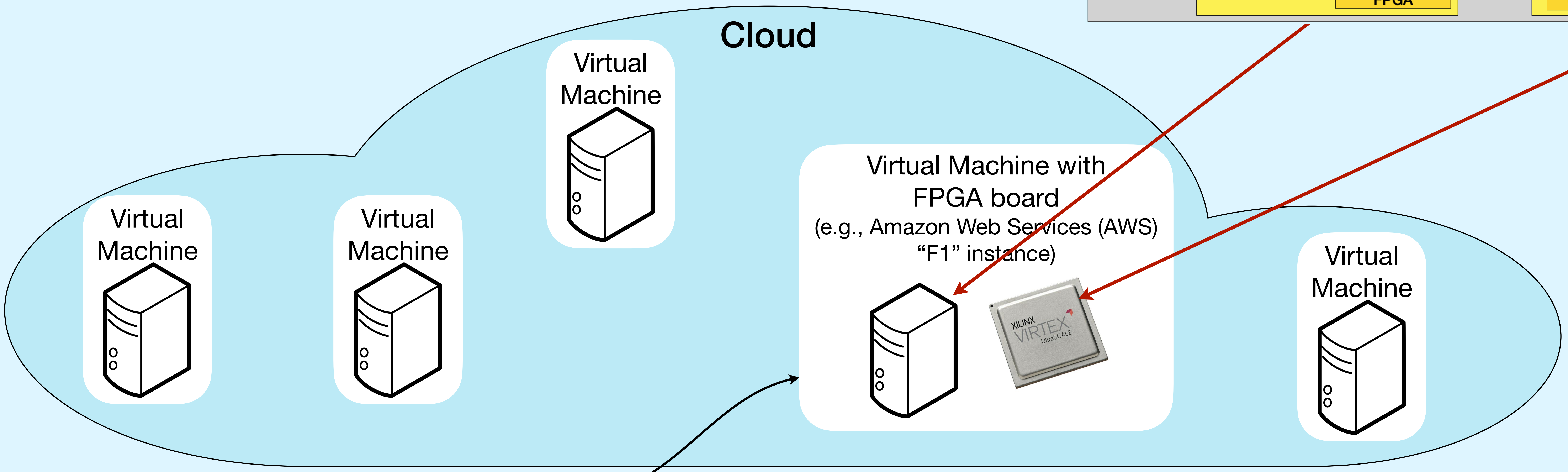
Infrastructure so Architect can focus on their creative work

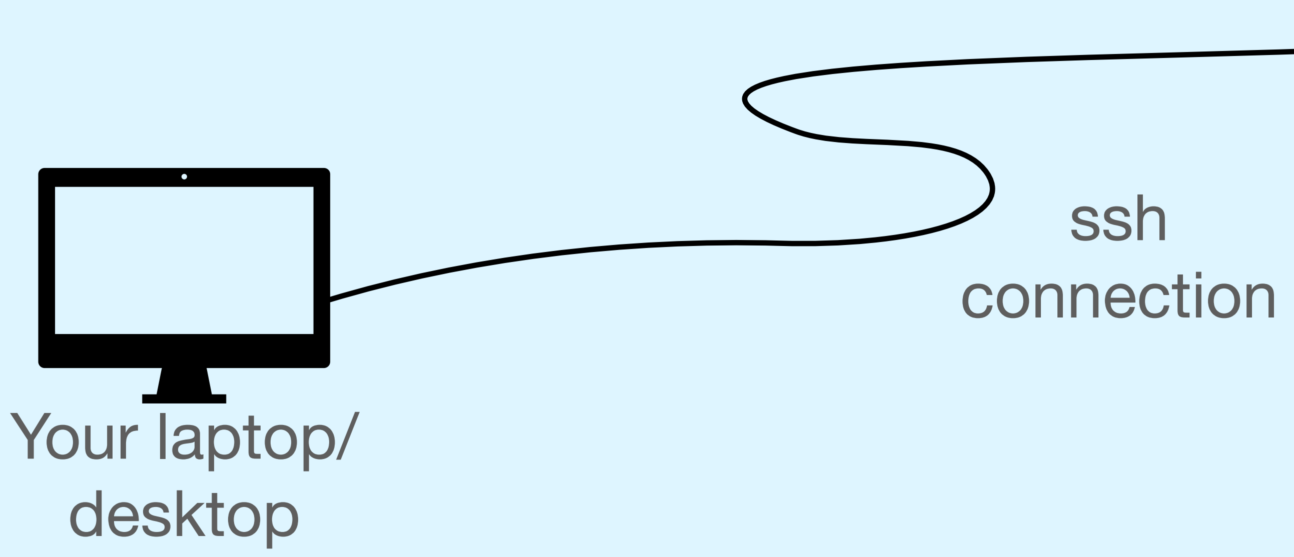


What is Catamaran/ARIFIC?

What are “FPGAs in the cloud”?







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

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.

Demos are live on Amazon AWS, subject to reasonable internet connectivity

Plan for this tutorial

Walk through: how to substitute your own core, and run it

- 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 …

Demos are live on Amazon AWS, subject to reasonable internet connectivity

Goal for this tutorial

By the end of this tutorial, you should feel confident you know:

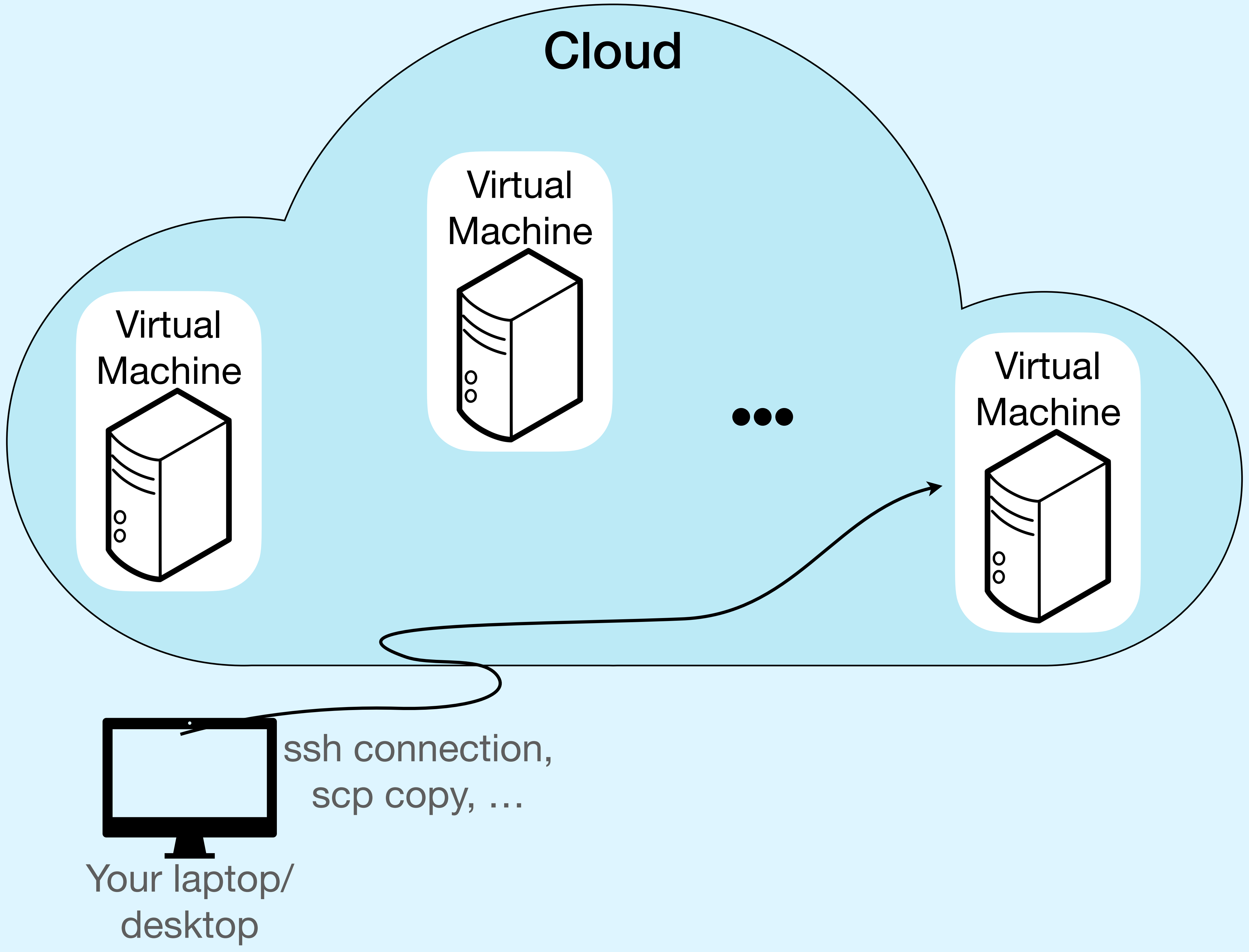
- How to open an AWS account and create AMIs (virtual machines) in which you can use Catamaran/ARIFIC
- How to download and install, into your AMIs, all the relevant software tools: AWS tools, RISC-V Gnu tools, Catamaran software and scripts
- How to replicate all the tutorial demos (compile and run ISA tests, compile and run bare-metal C programs, boot Linux, use networking and block device in Linux)
  - all using the pre-built AWS bitfiles (with Catamaran+Rocket and Catamaran+CVA6)
- How to substitute your own core in place of Rocket/CVA6, create a new bitfile, load it, debug it, run code on it, measure it.

Basics of Amazon AWS

- *AMIs (virtual machines in the Amazon cloud)*
- Connecting to an AMI using “ssh”
- About AWS bitfiles; available Catamaran/ARIFIC bitfiles

Basics of Amazon AWS

AMIs (virtual machines in Amazon cloud)



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.

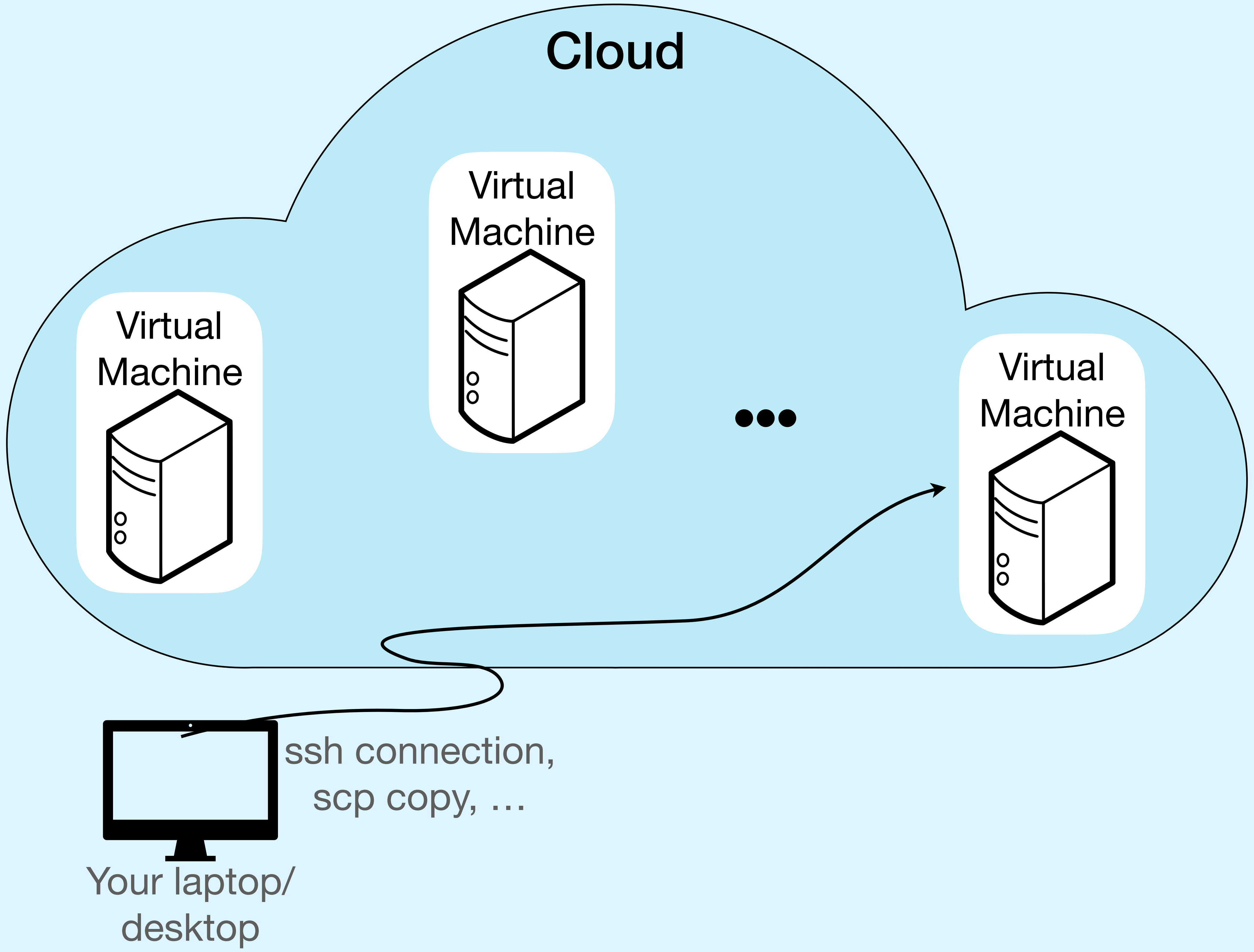
Costs:

- 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.

Amazon terminology for a virtual machine in their cloud:
“AMI” = “Amazon Machine Instance”
We also just say “instance” for an AMI/virtual machine

Basics of Amazon AWS

AMIs (virtual machines in Amazon cloud)



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; available Catamaran/ARIFIC bitfiles

*Disclaimer: this **not** official Amazon documentation, and is only based on this author's observations.*

Basics of Amazon AWS

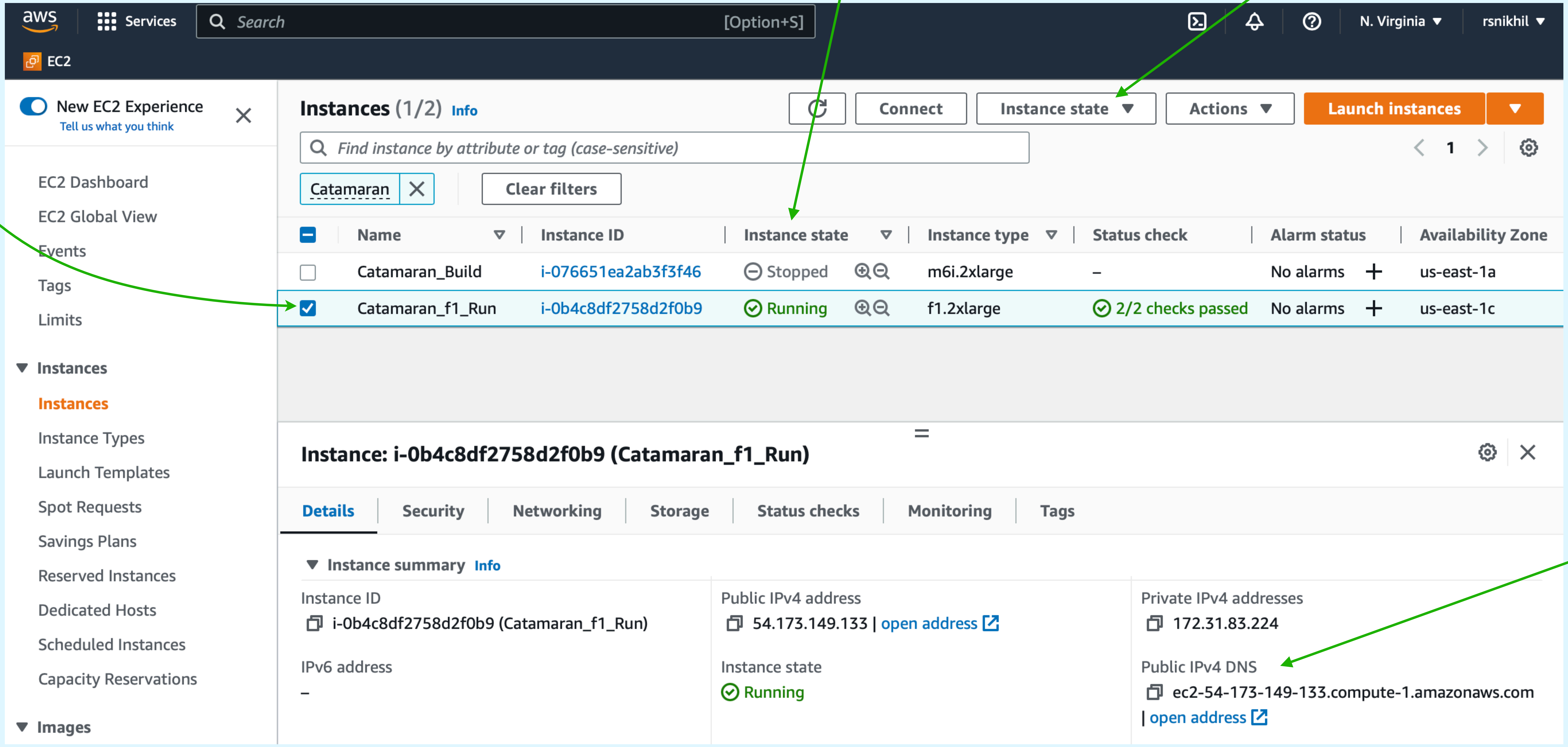
Connecting to your AMI with ssh (1/2)

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

*Disclaimer: this **not** official Amazon documentation, and is only based on this author's observations.*

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
```

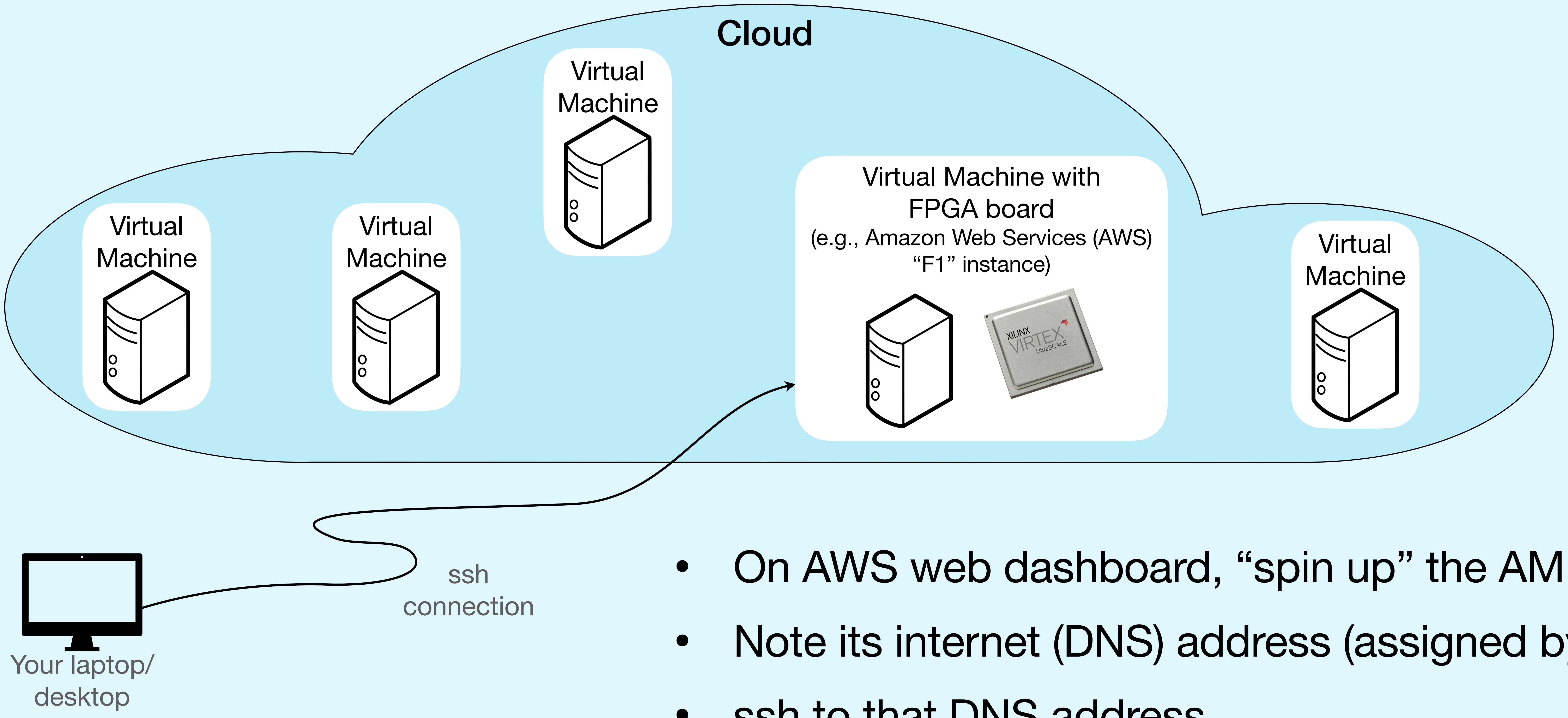
Note: you can copy files to and from your AMI using "scp" (which is based on "ssh")

```
$ 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; available Catamaran/ARIFIC 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?)

Basics of Amazon AWS

Available pre-built bitfiles in Catamaran/ARIFIC (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 (expected 3/2023)

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

Basics of Amazon AWS

HDL source for pre-built bitfiles in Catamaran/ARIFIC (all based on open-source CPUs)

CPU	*Original HDL*	
Berkeley/SiFive Rocket	Chisel	https://github.com/chipsalliance/rocket-chip
Bluespec Flute	BSV	https://github.com/bluespec/Flute
OpenHardware Group CVA6 (a.k.a. ETH Zurich Ariane)	SystemVerilog	https://github.com/openhwgroup/cva6.git

Note: all three are actually CPU "generators": each can be parameterized to generate a RISC-V CPU from as small as RV32I to as large as RV64IMAFDC + optional MSU privilege + optional Sv32/Sv39 virtual memory.
Microarchitectures can also be parameterized (organization/size of cache/MMU, branch predictors, …).
Any such configuration can be plugged into Catamaran/ARIFIC.

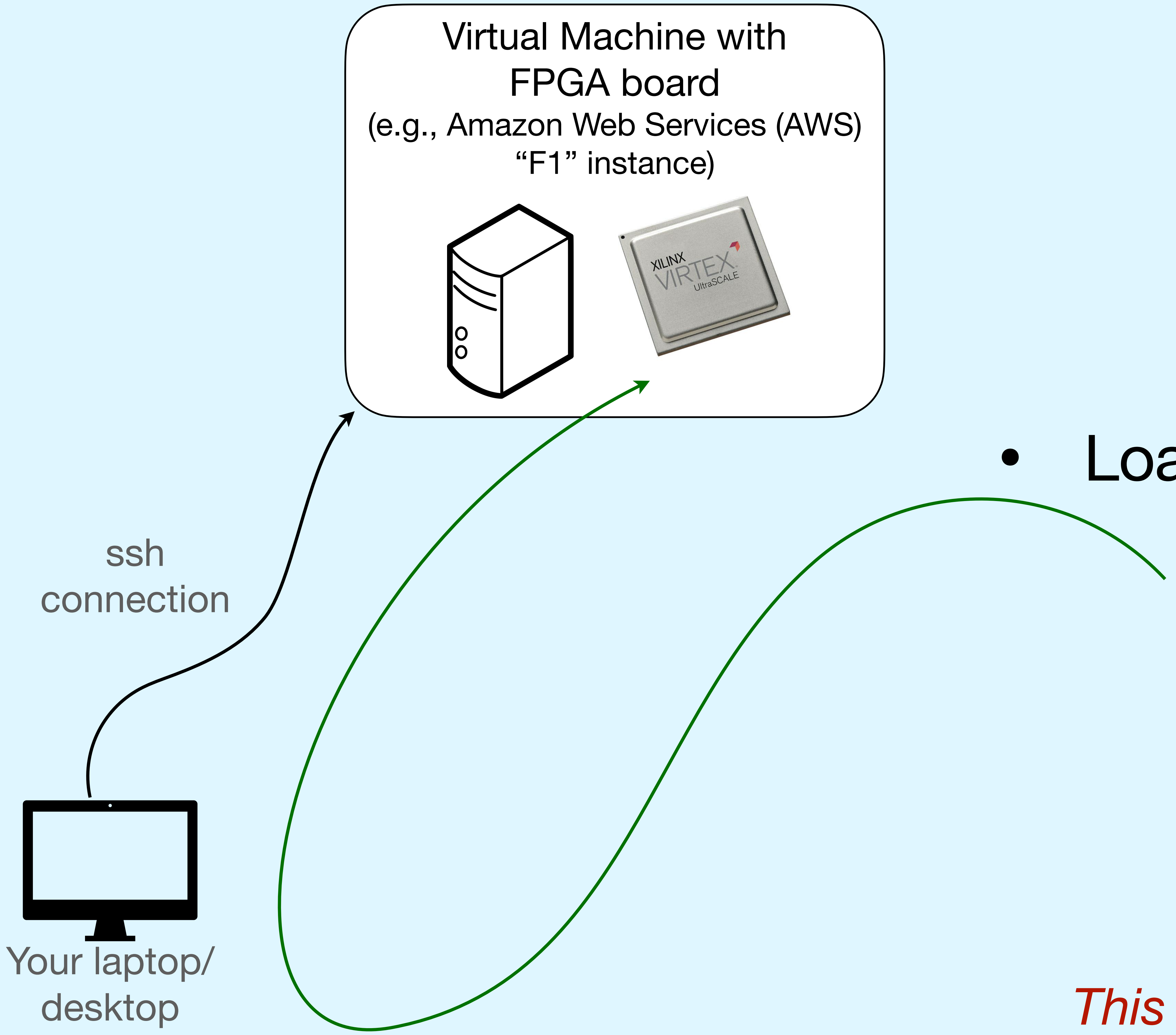
Other low-hanging fruit to fit into Catamaran/ARIFIC:

- Berkeley/SiFive BOOM (high-end speculative, superscalar, out-of-order)
- Bluespec Magritte, Piccolo (small, embedded class)
- Bluespec Toooba, based on MIT RISCY-OOO (high-end speculative, superscalar, out-of-order)

One of these may be a good baseline starting point for your architecture research!

Basics of Amazon AWS

DEMO: Loading an AWS bitfile into the FPGA



- 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*

What you can do with Catamaran/ARIFIC

- *DEMO: Cross-compile and run ISA tests*
- DEMO: Cross-compile C programs for bare-metal and Linux
- DEMO: Run C programs on bare-metal
- DEMO: Run Linux, with networking and block devices
- DEMO: Run C programs under Linux

What you can do with Catamaran/ARIFIC

Cross-compile and 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

Cross-compile ISA tests

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

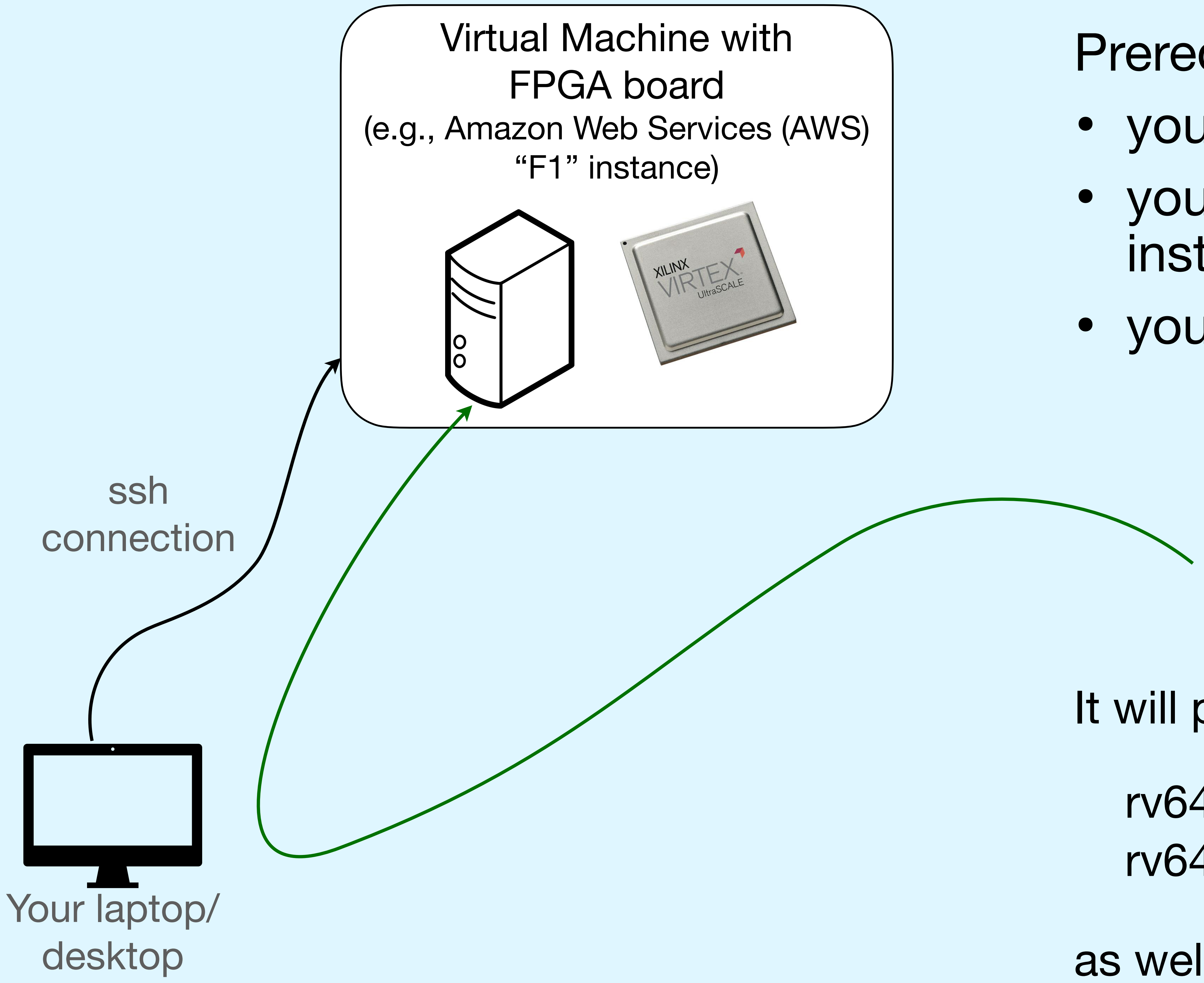
This tutorial repository contains, in the dir ISA_Tests/

- 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

What you can do with Catamaran/ARIFIC

DEMO: cross-compile RISC-V ISA tests



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:

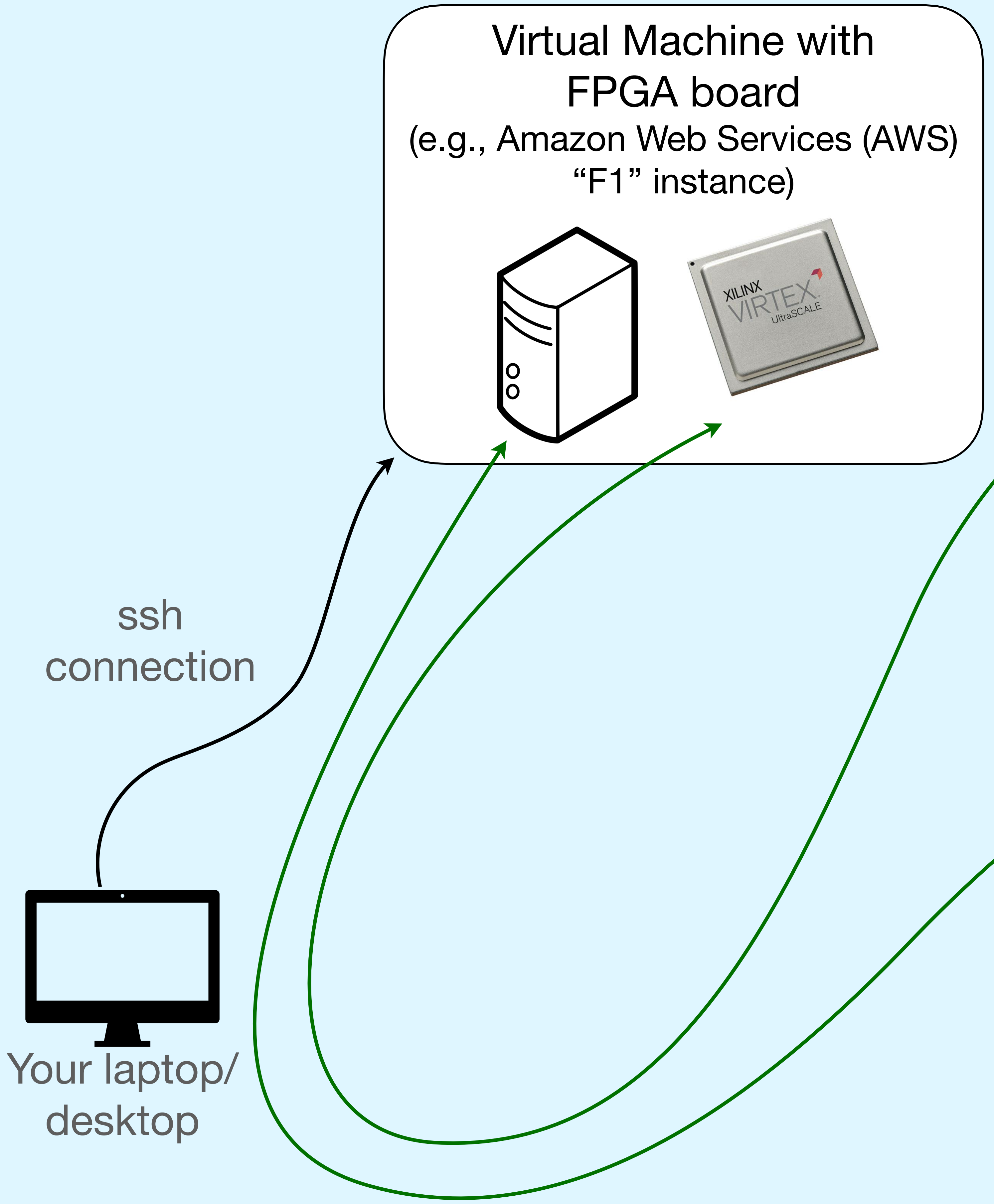
rv64ui-p-add	(user-level, integer, physical memory, ‘add’ opcode)
rv64ui-v-add	(…, virtual memory, …)

as well as “object dump” (disassembly) files likes this:

rv64ui-p-add.dump

What you can do with Catamaran/ARIFIC

DEMO: run RISC-V ISA tests



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  ~/Git/Tutorial_at_HPCA-29/ISA_Tests/isa
$ /opt/bluespec/catamaran/bin/exe_Host_ISA_Tester_AWSF1  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)

What you can do with Catamaran/ARIFIC

- DEMO: Cross-compile and run ISA tests
- *DEMO: Cross-compile C programs for bare-metal and Linux*
- DEMO: Run C programs on bare-metal
- DEMO: Run Linux, with networking and block devices
- DEMO: Run C programs under Linux

What you can do with Catamaran/ARIFIC

Bare-metal (no OS) C programs and C programs under Linux

Bare-metal C program	C program under Linux
No system calls. E.g., “printf/putc/putchar”, “scanf/getc/getchar” are compiled to function calls to library code that directly interacts with a console device.	Has system calls for OS services. E.g., “printf/putc/putchar”, “scanf/getc/getchar” are compiled to system calls, and the OS interacts with a console device.
No “exit”; typically ends in an infinite idle loop.	Typically ends in an “exit” system call that reverts to the OS scheduling loop.

What you can do with Catamaran/ARIFIC

Two small example C programs in the repository

This tutorial repository contains

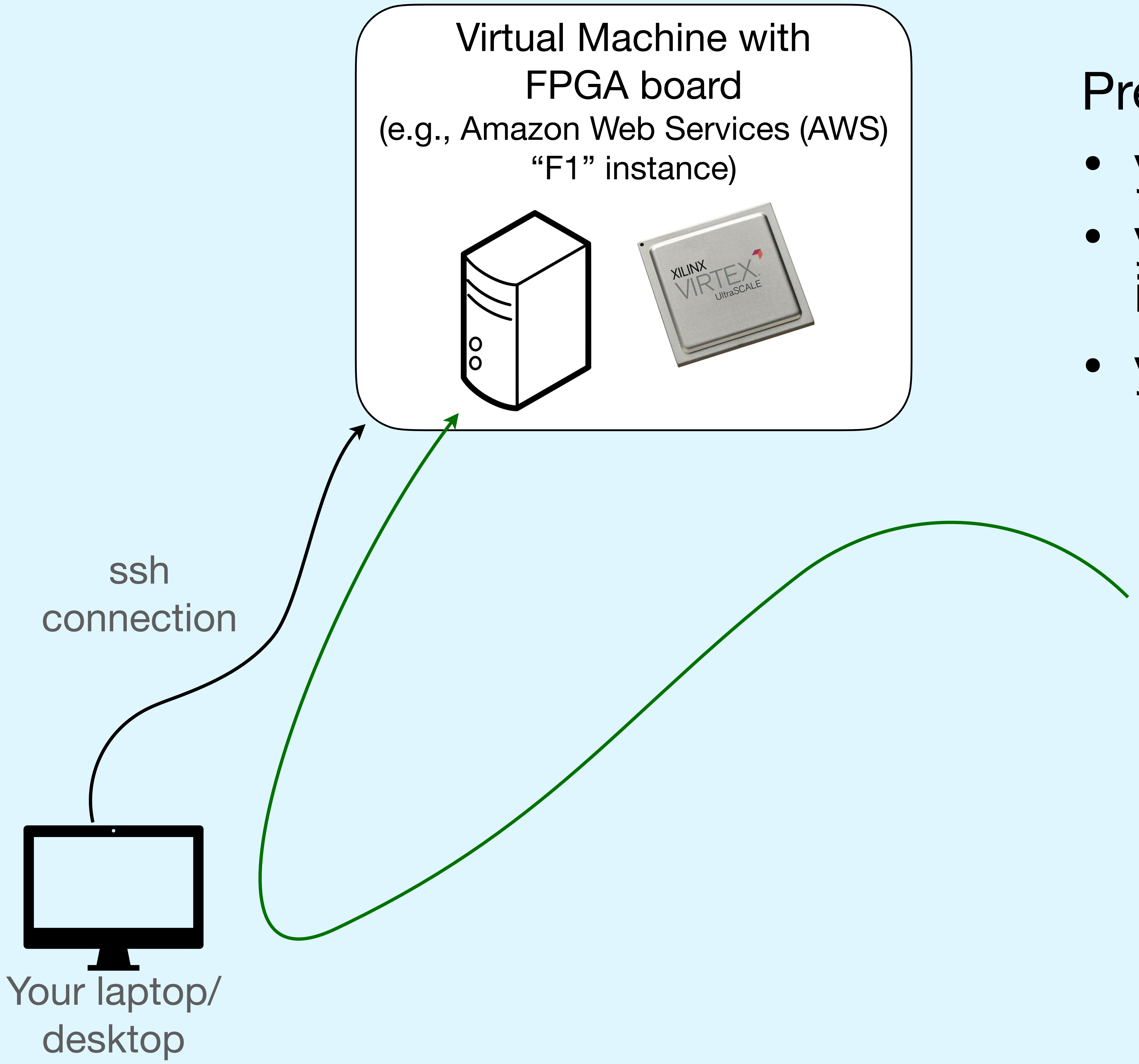
- Two small C programs: “Hello World!” and “cat” (echoes stdin to stdout)
- A Makefile and linker scripts to cross-compile the C programs for RV64GC
- Pre-built results of cross-compiling the examples: two RISC-V ELF files

Note: our linker script sets

- Catamaran/ARIFIC program-start address = 0x_8000_0000
- UART MMIO address = 0x_6010_0000

What you can do with Catamaran/ARIFIC

DEMO: Cross-compile C programs to run on “bare-metal” (no OS)



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
```

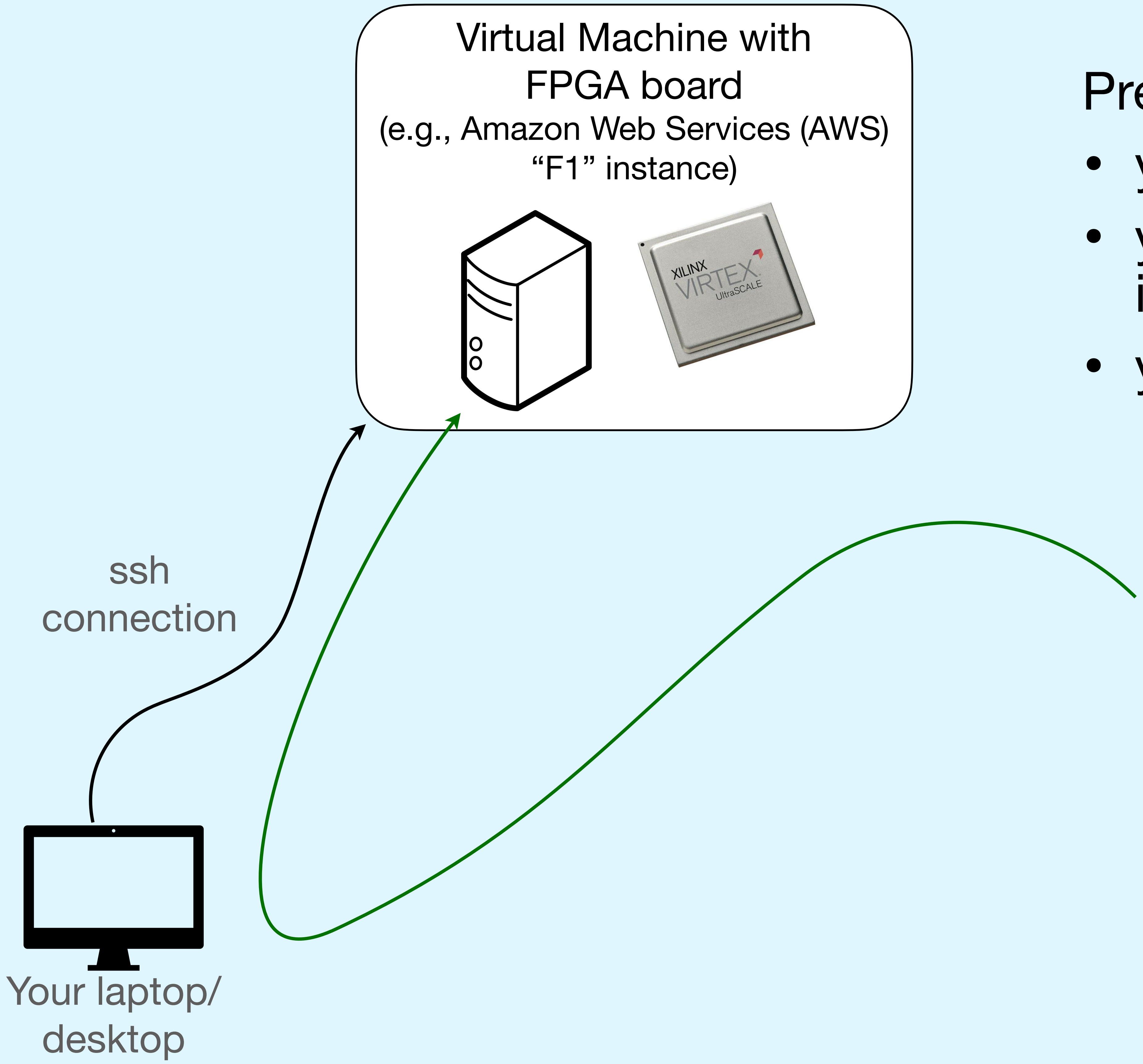
Creates: hello.RV64.bare.elf hello.RV64.bare.map hello.RV64.bare.objdump

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

```
$ cd  C_Examples/cat
$ make  all_bare
```

What you can do with Catamaran/ARIFIC

DEMO: Cross-compile C programs to run under Linux



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
```

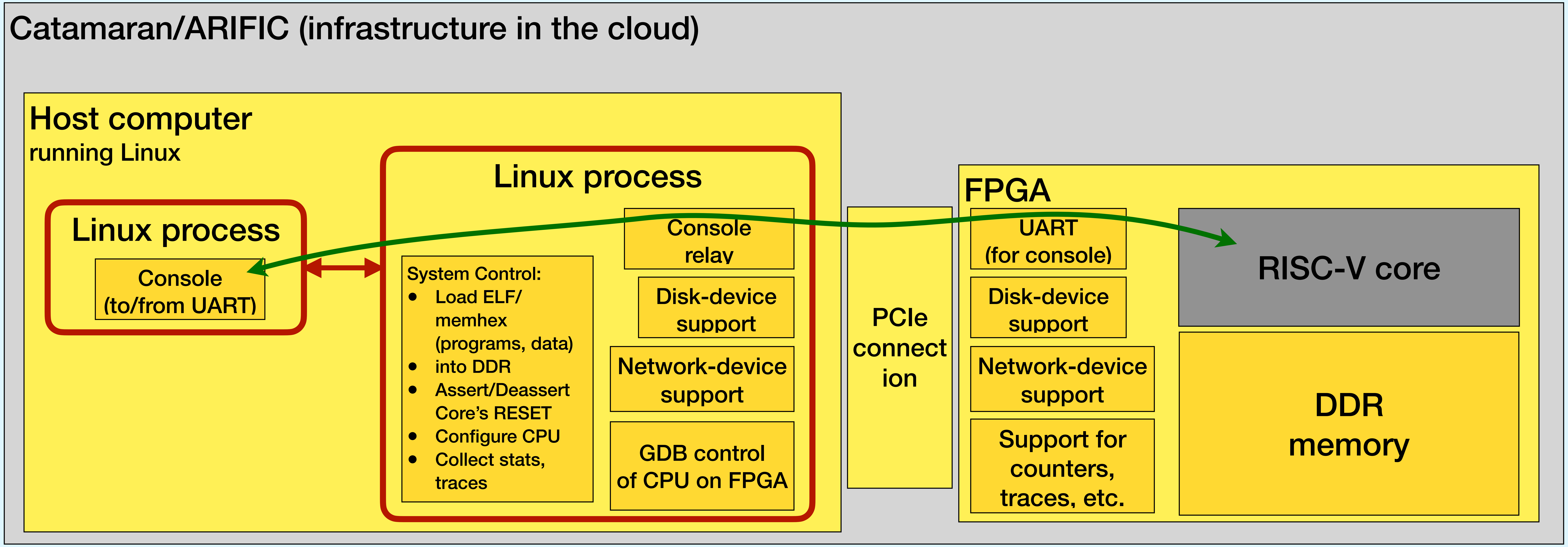
What you can do with Catamaran/ARIFIC

- DEMO: Cross-compile and run ISA tests
- DEMO: Cross-compile C programs for bare-metal and Linux
- *DEMO: Run C programs on bare-metal*
- DEMO: Run Linux, with networking and block devices
- DEMO: Run C programs under Linux

What you can do with Catamaran/ARIFIC

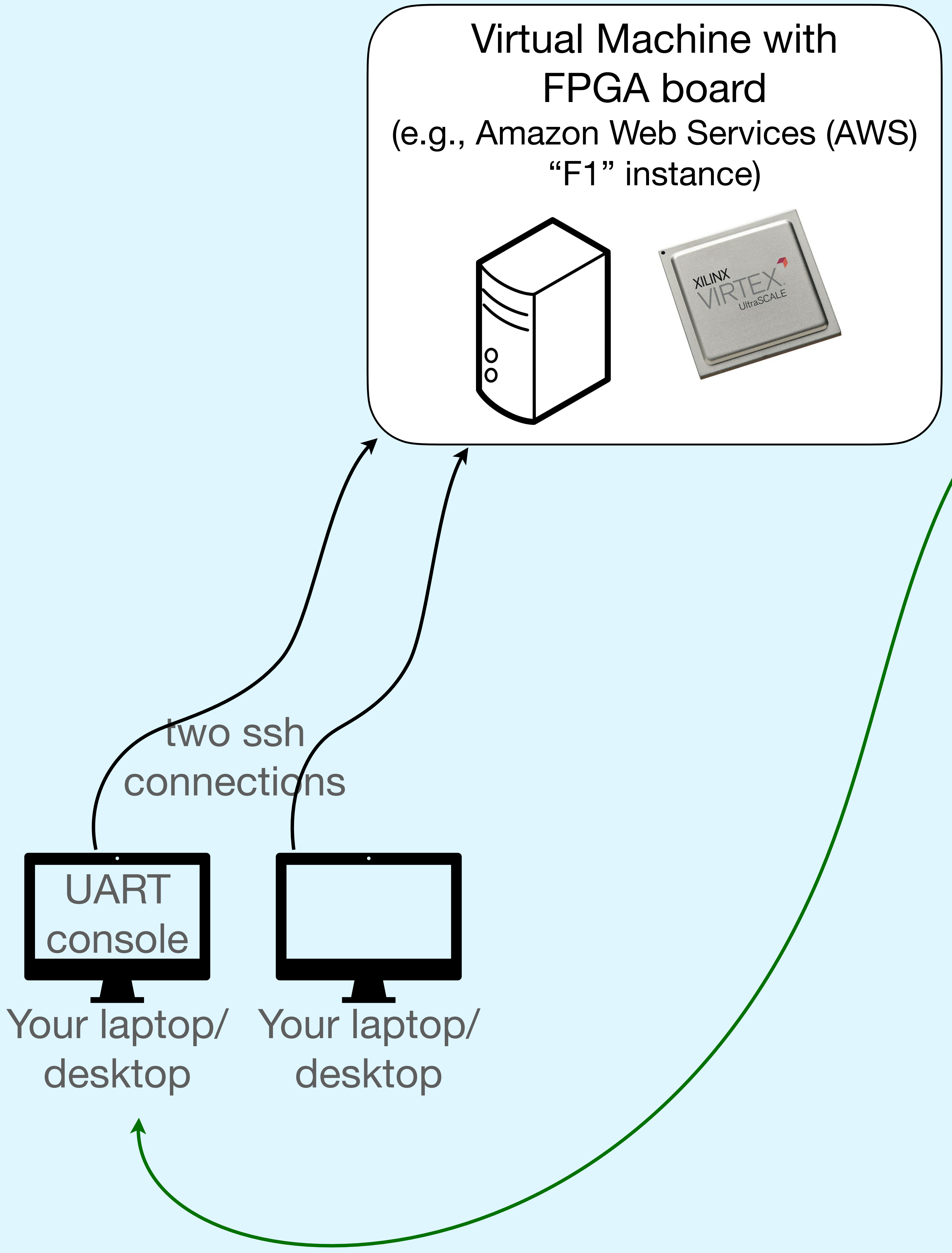
For most programs we need a UART (console device for “printf”)

- Catamaran/ARIFIC contains a UART and connections to a terminal on the host computer



What you can do with Catamaran/ARIFIC

DEMO: Open a second terminal on the AMI for the UART process



Make 2nd ssh connection to AMI; start UART console

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

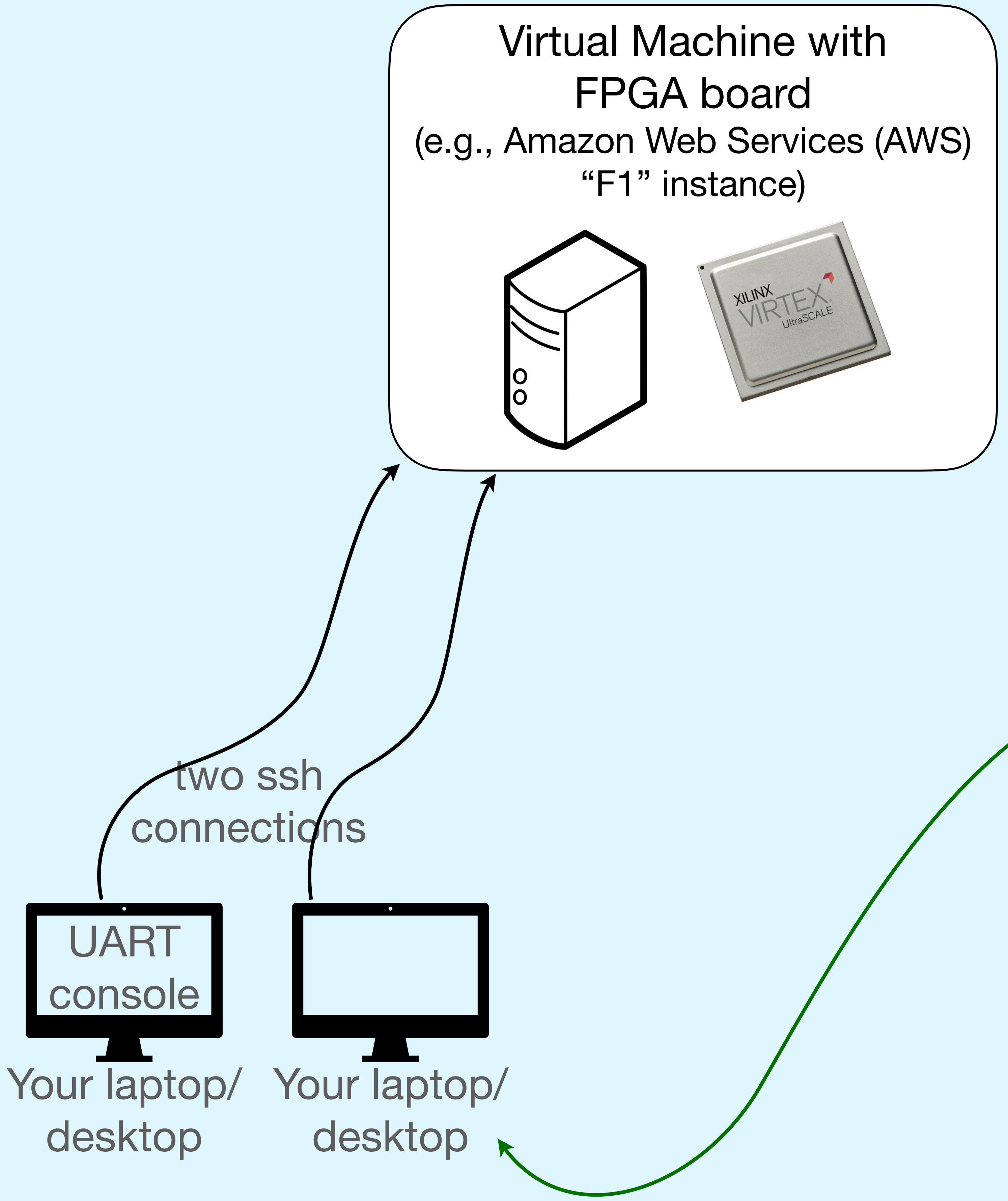
In the terminal, start the “UART_Console” program

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

```
--03:54:47--ip-172-31-83-224: ~/Git/Tutorial_at_HPCA-29/Host_side_exes
$ sudo ./UART_Console --log log_UART.txt
ptyname_file filename = UART_ptyname_file
UART output logfile name = log_UART.txt
Awaiting file 'UART_ptyname_file' for reading pty name
```

What you can do with Catamaran/ARIFIC

DEMO: run bare-metal C programs



(Assume FPGA bitfile loaded from previous activity.)

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

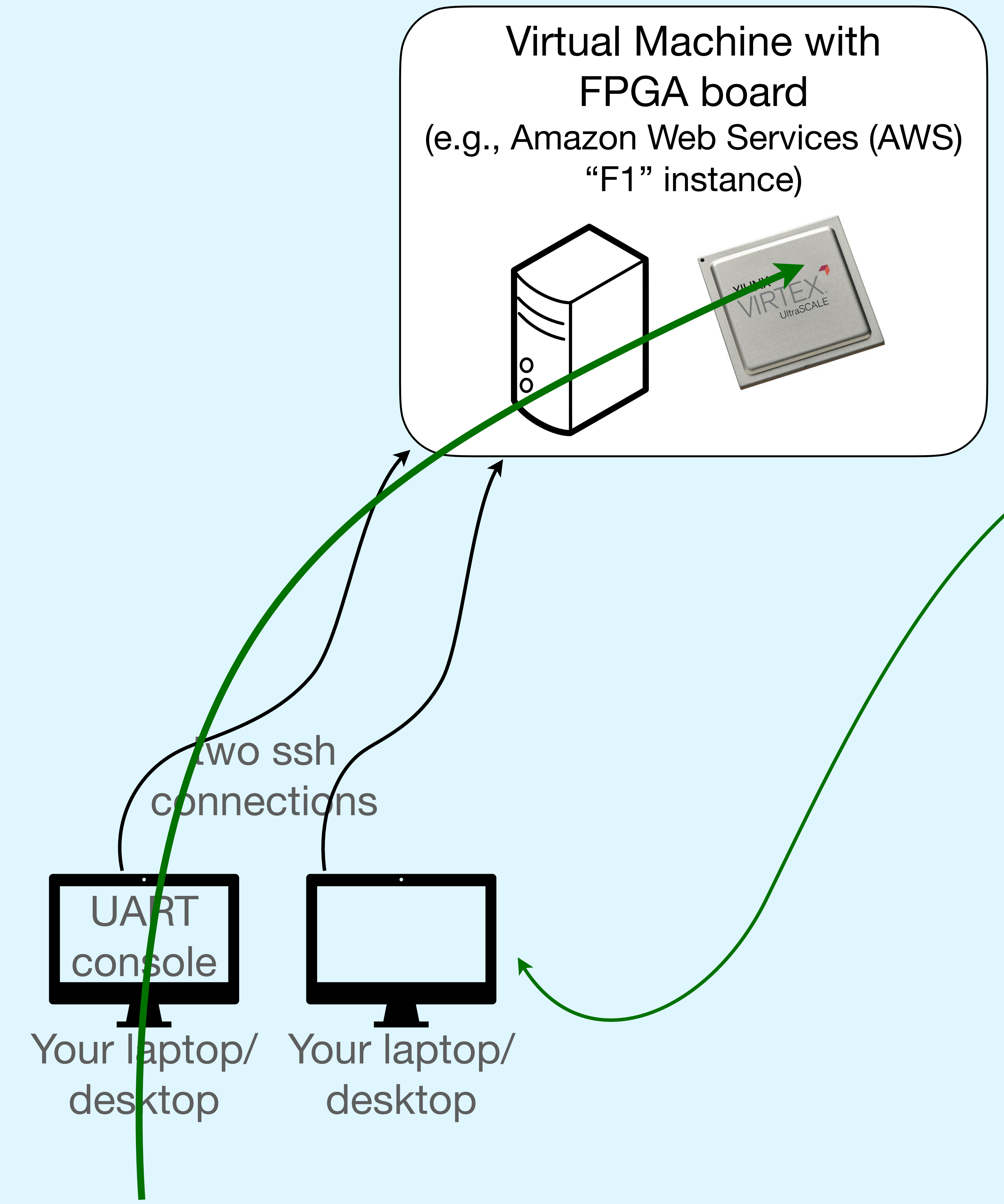
… observe "Hello World!" on UART console …

What you can do with Catamaran/ARIFIC

- DEMO: Cross-compile and run ISA tests
- DEMO: Cross-compile C programs for bare-metal and Linux
- DEMO: Run C programs on bare-metal
- *DEMO: Run Linux, with networking and block devices*
- DEMO: Run C programs under Linux

What you can do with Catamaran/ARIFIC

DEMO: Run Linux



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
```

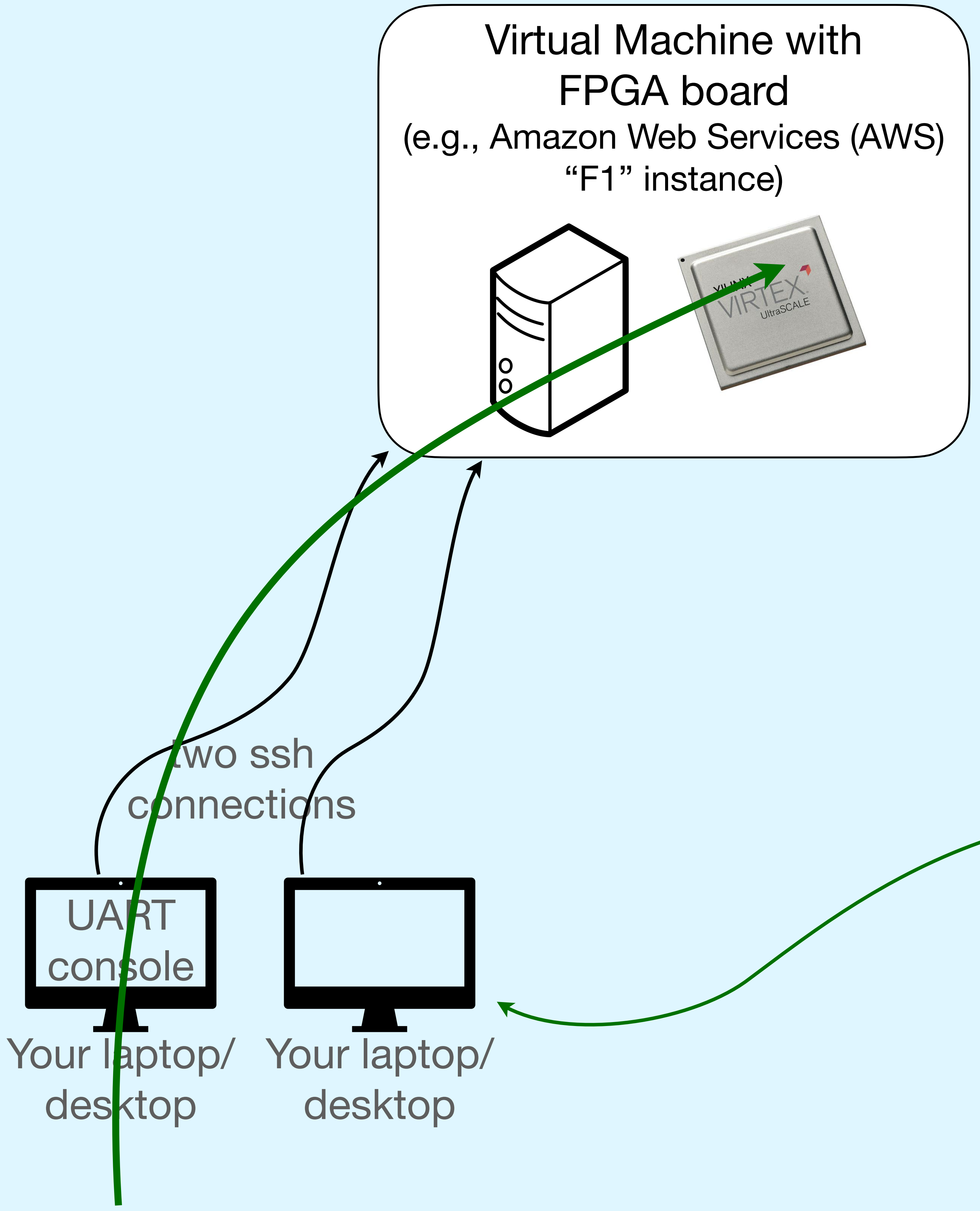
Observe console messages from OpenSBI boot loader,
followed by console messages from Linux boot,
finally the Linux shell prompt.
Then, …

… interact with the Linux shell

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)



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.

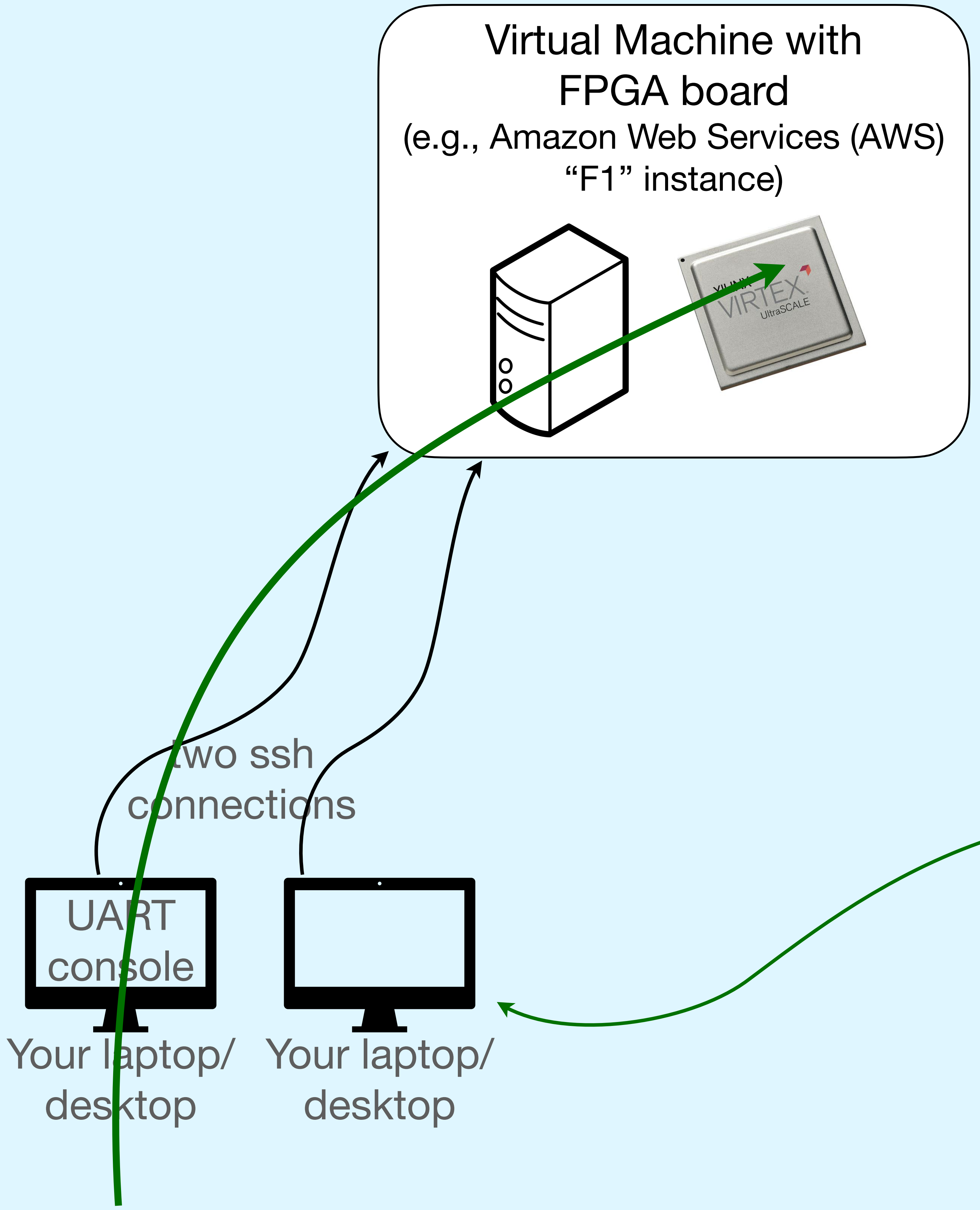
Boot Linux again on the FPGA, providing it these facilities:

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

… 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)



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
```

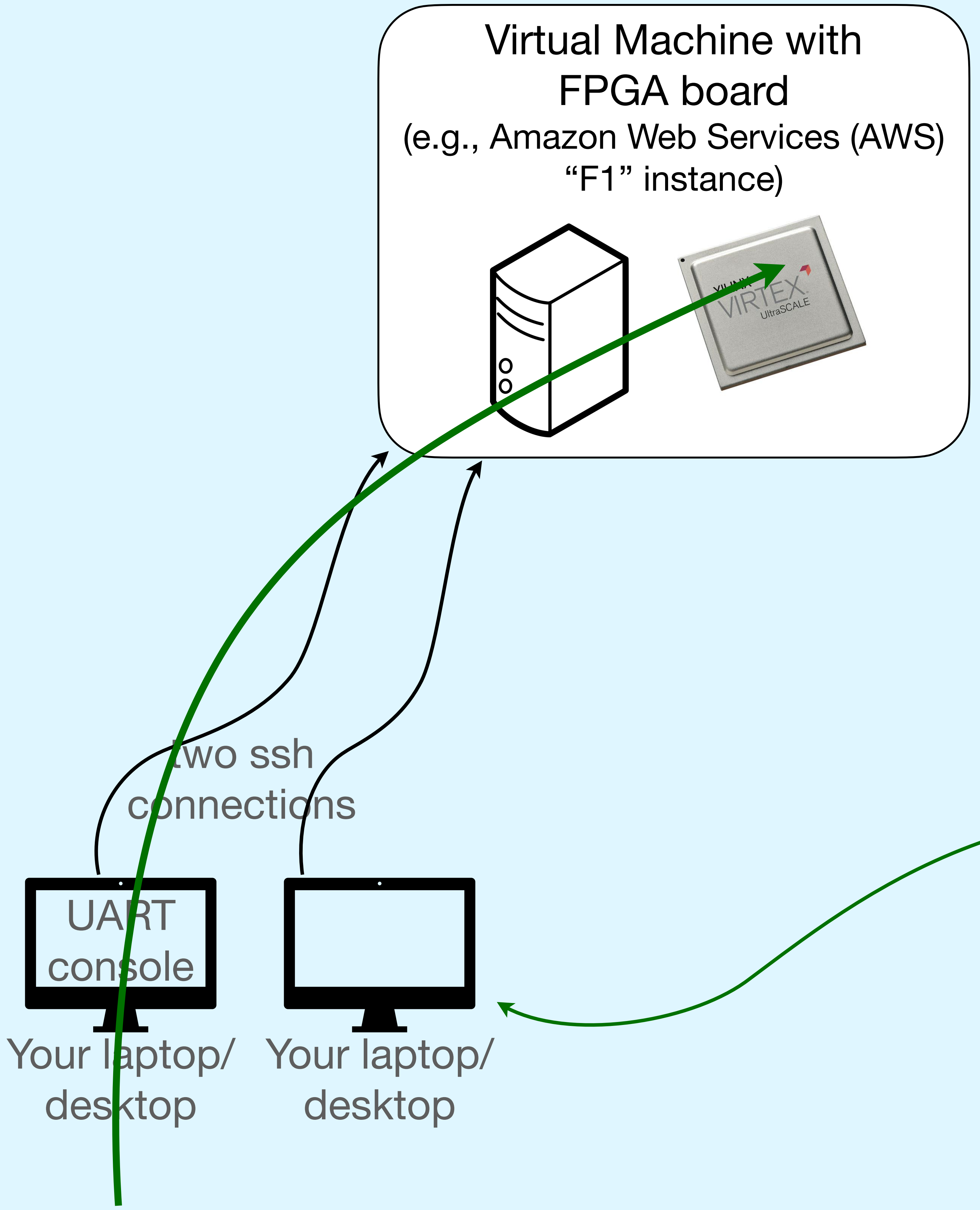
Interact with the filesystem

```
# ls ..
# cat …
```

… interact with the Linux shell and use block device

What you can do with Catamaran/ARIFIC

DEMO: Run Linux with a network and block device (1/2)



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.

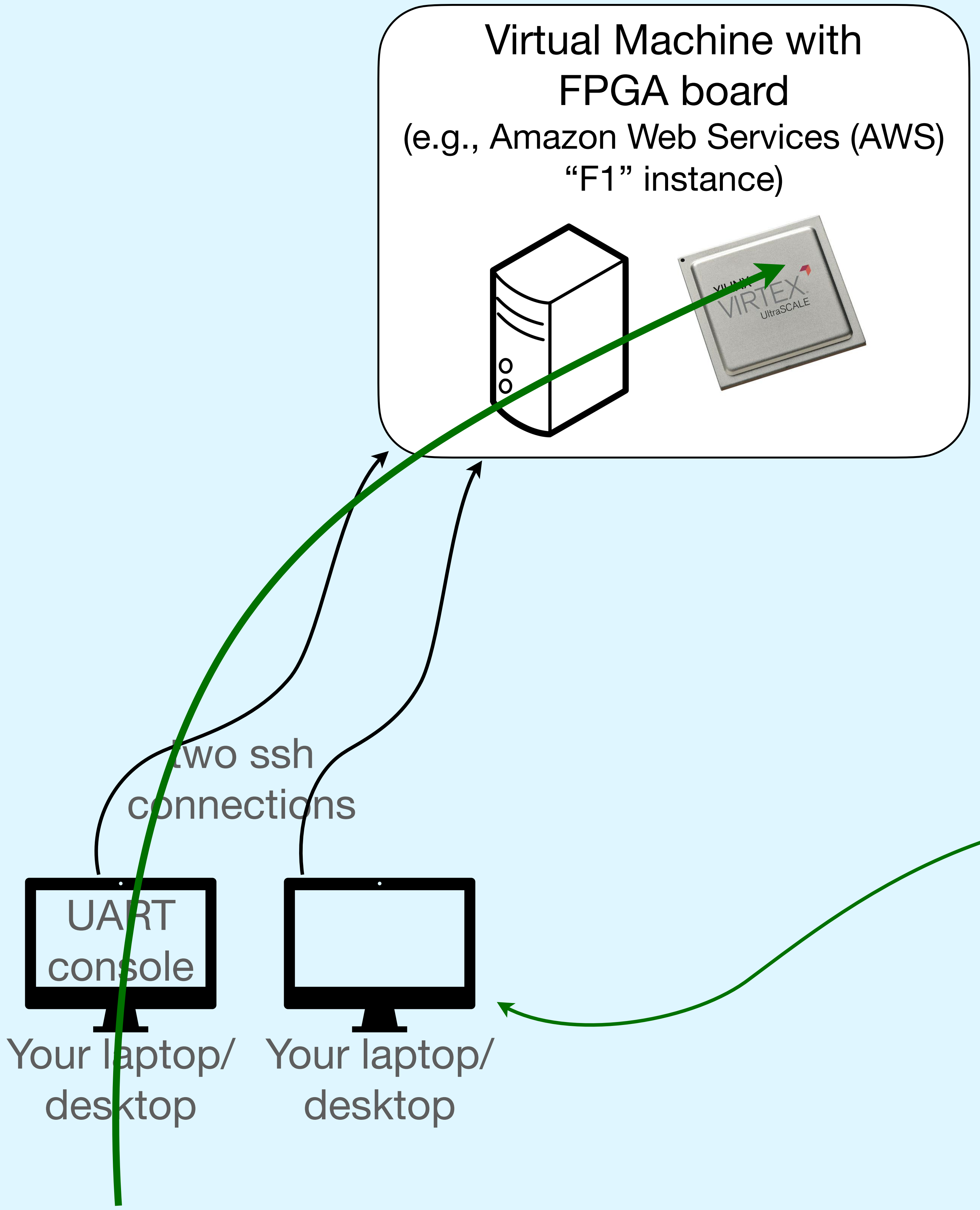
Boot Linux again on the FPGA, providing it these facilities:

```
$ 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)



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
```

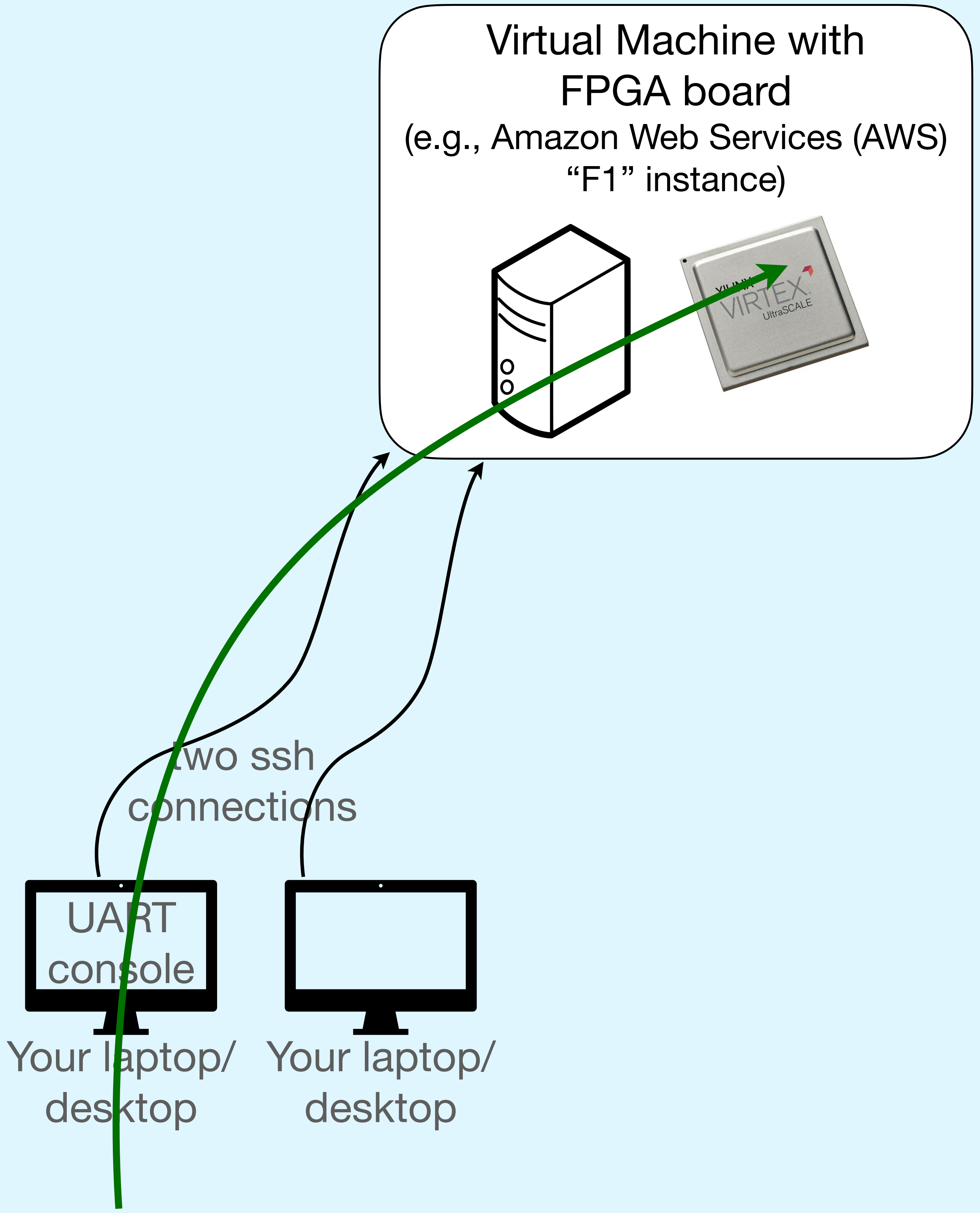
… interact with the Linux shell and use network and block device

What you can do with Catamaran/ARIFIC

- DEMO: Cross-compile and run ISA tests
- DEMO: Cross-compile C programs for bare-metal and Linux
- DEMO: Run C programs on bare-metal
- DEMO: Run Linux, with networking and block devices
- *DEMO: Run C programs under Linux*

What you can do with Catamaran/ARIFIC

DEMO: Run C programs under Linux



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, run C programs under Linux

What you can do with Catamaran/ARIFIC

- DEMO: Cross-compile and run ISA tests
- DEMO: Cross-compile C programs for bare-metal and Linux
- DEMO: Run C programs on bare-metal
- DEMO: Run Linux, with networking and block devices
- DEMO: Run C programs under Linux
- *Collecting measurement data*

What you can do with Catamaran/ARIFIC

Collecting measurement data

“Hardware Performance Monitor” counters: The RISC-V Privileged ISA Spec defines 29 64-bit performance counters that can be read/written as CSRs (Control/Status Registers). The functionality of these counters (what events they count) is left up to each platform (each core).

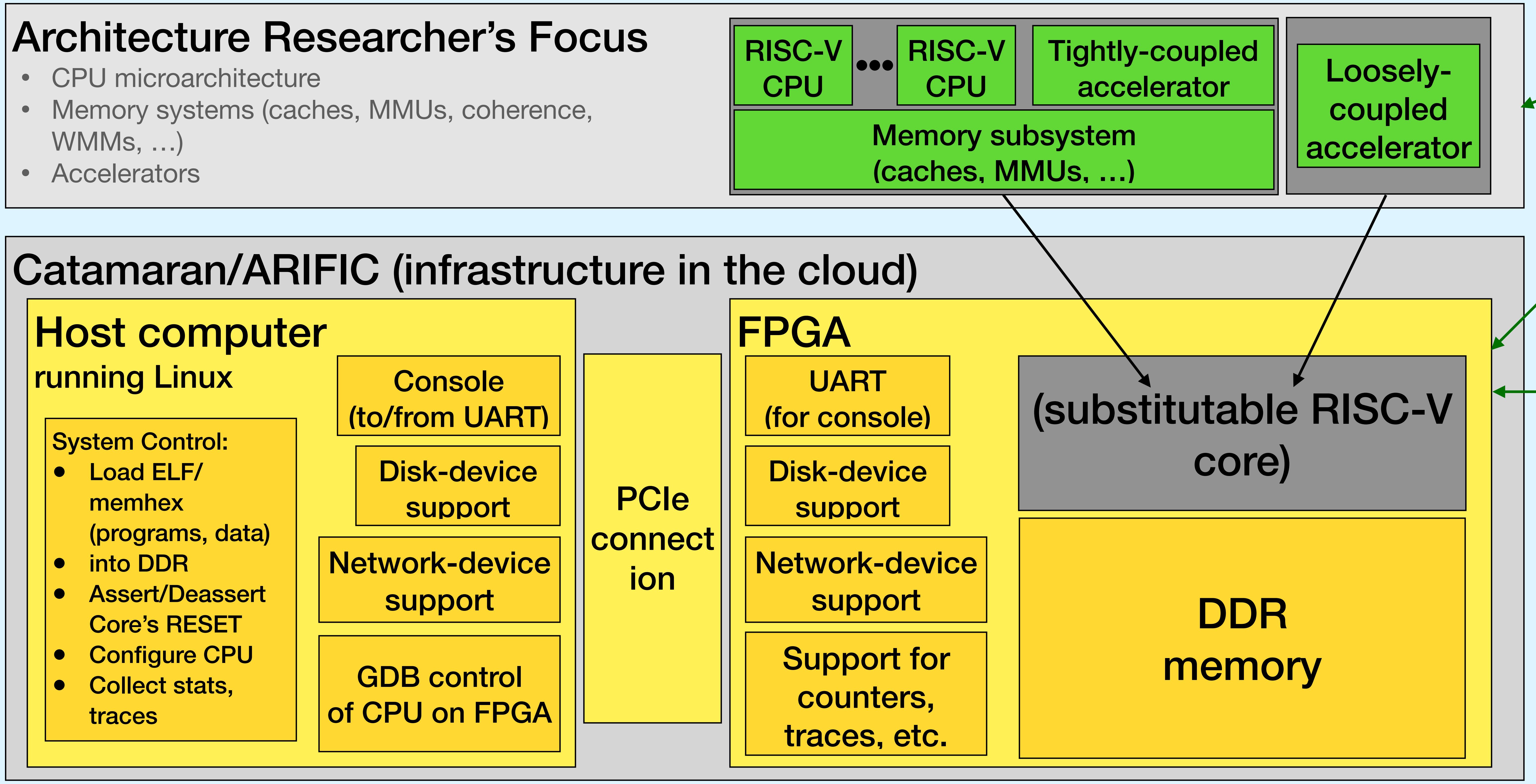
During, and at the end of an experiment, code can record hpmcounter values to memory.
The Host-Side Hub’s interactive prompt has a command to dump a DDR memory region to a file.

Note: the AWS F1 FPGA board has 64 GiB of DDR memory.

Plug in your own RISC-V Core

- The Catamaran/ARIFIC Core Interface (RTL)
- DEMO: Build and run a whole-system simulation executable
- DEMO: Build an FPGA bitfile for Amazon AWS

Plug in your own RISC-V core: goals



Assume you've created your own core[1]

1. Build a whole-system simulation executable of the FPGA-side, with your core substituted in place of the demo cores
2. Build the FPGA bitfile of the FPGA-side, with your core substituted in place of the demo cores, and run on AWS

[1] "Your own RISC-V Core"

Options:

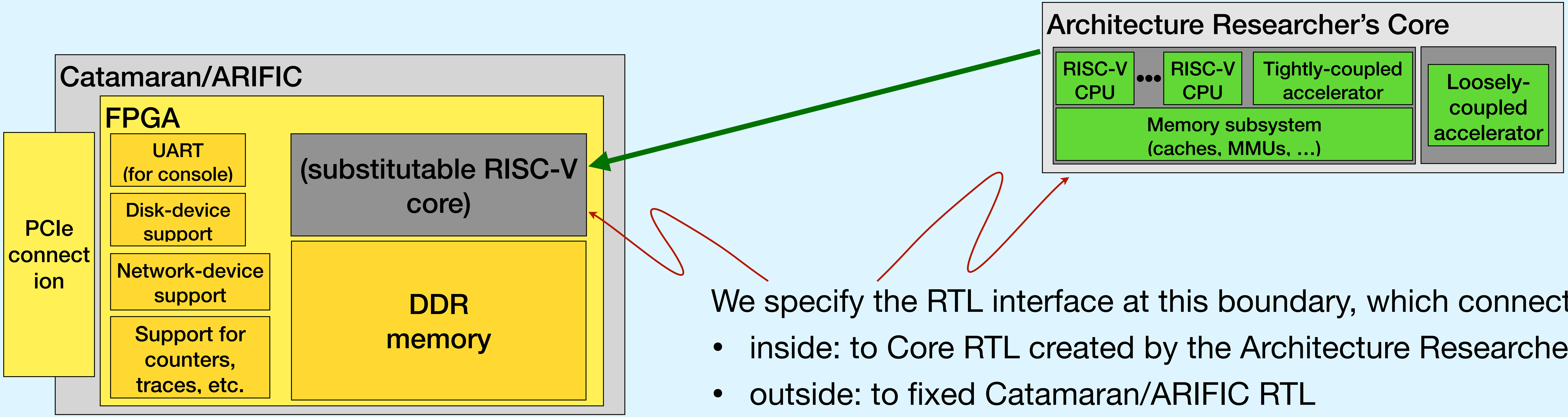
- Your own new CPU design
- Modify available CPU (many open-source)
  - microarchitecture change
  - new instruction
  - new CSRs (e.g., counters)
- Modify/replace memory system (caches, MMUs, PMPs, PTWs, …)
- Add tightly-coupled or loosely coupled accelerator
- … or other research idea …

Plug in your own RISC-V Core

- *The Catamaran/ARIFIC Core Interface (RTL)*
- DEMO: Build and run a whole-system simulation executable
- DEMO: Build an FPGA bitfile for Amazon AWS

Plug in your own RISC-V core

The Catamaran/ARIFIC Core Interface (RTL)



We specify the RTL interface at this boundary, which connects

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

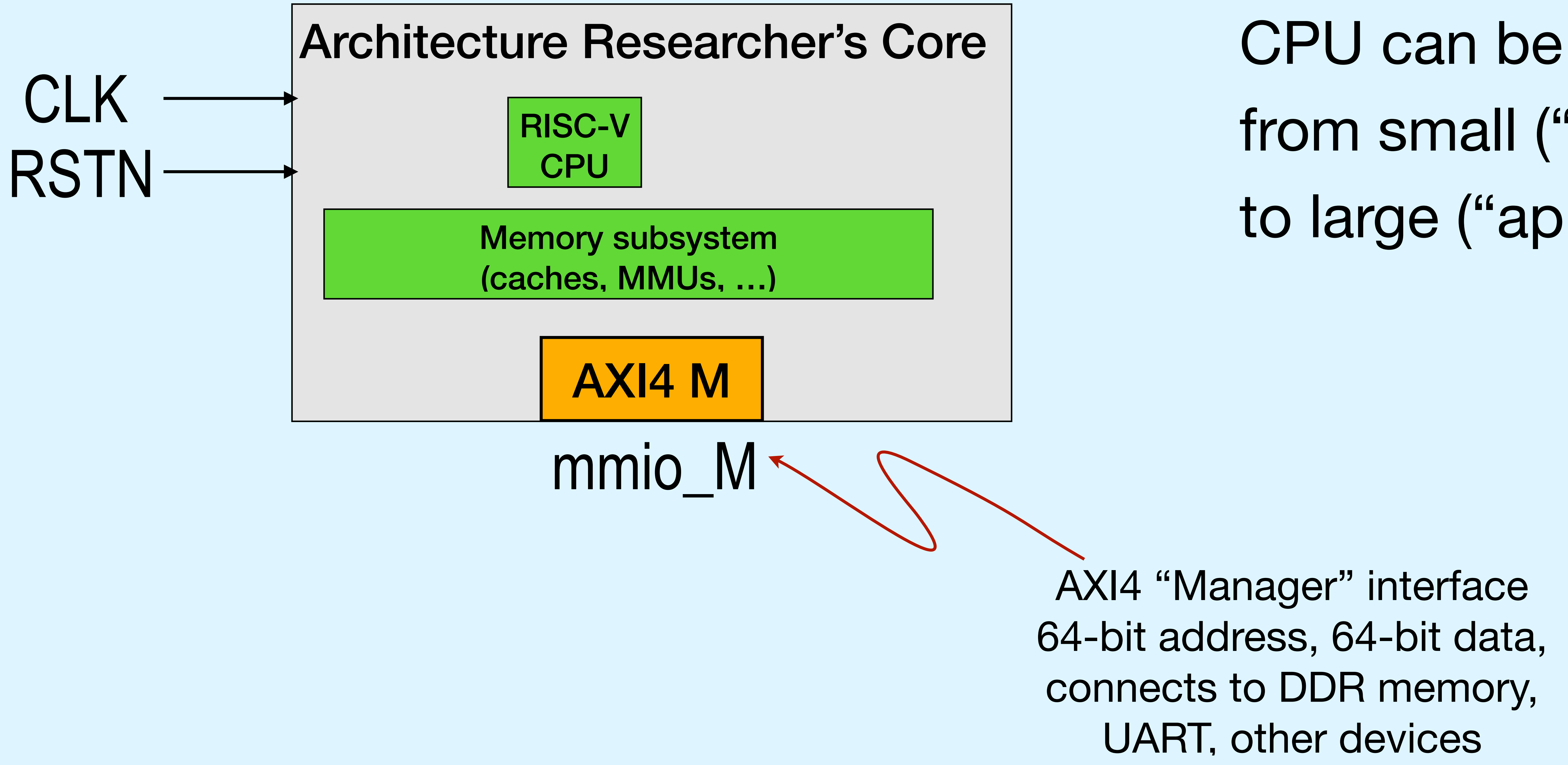
Please see this file in the tutorial repository: Catamaran_Core_Interface/mkAWSteria_Core_EMPTY.v
which contains a template "empty" core module with the required RTL inputs and outputs.
The Architecture Researcher substitutes their core for the module body, connects inputs/outputs.

Plug in your own RISC-V core

The Catamaran/ARIFIC Core Interface (RTL)

DEMO: View actual interface RTL in tutorial repository: Catamaran_Core_Interface/mkAWSteria_Core_EMPTY.v

Minimal core



CPU can be RV32 or RV64, from small ("embedded, IoT") to large ("application", "server").

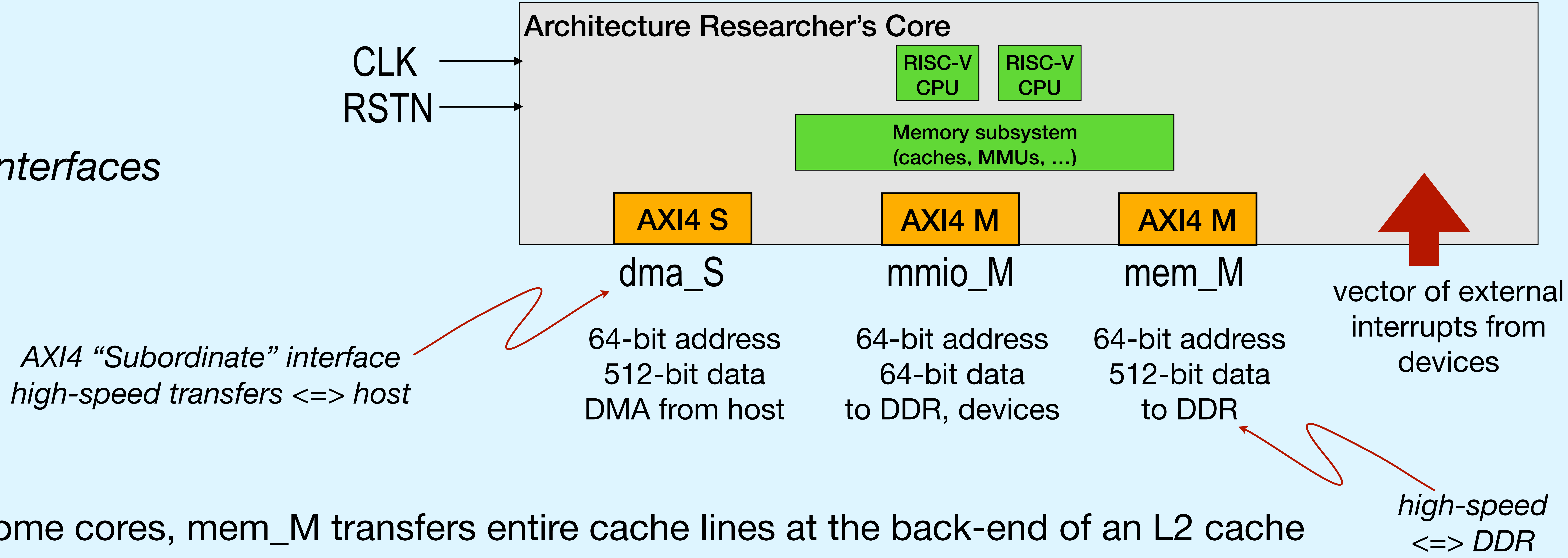
CLK can connect to one of six available clocks provided by Catamaran/ARIFIC, ranging from about 15 MHz to 250 MHz, depending on the speed/timing requirements of the core.

Plug in your own RISC-V core

The Catamaran/ARIFIC Core Interface (RTL)

DEMO: View actual interface RTL in tutorial repository: Catamaran_Core_Interface/mkAWSteria_Core_EMPTY.v

More interfaces



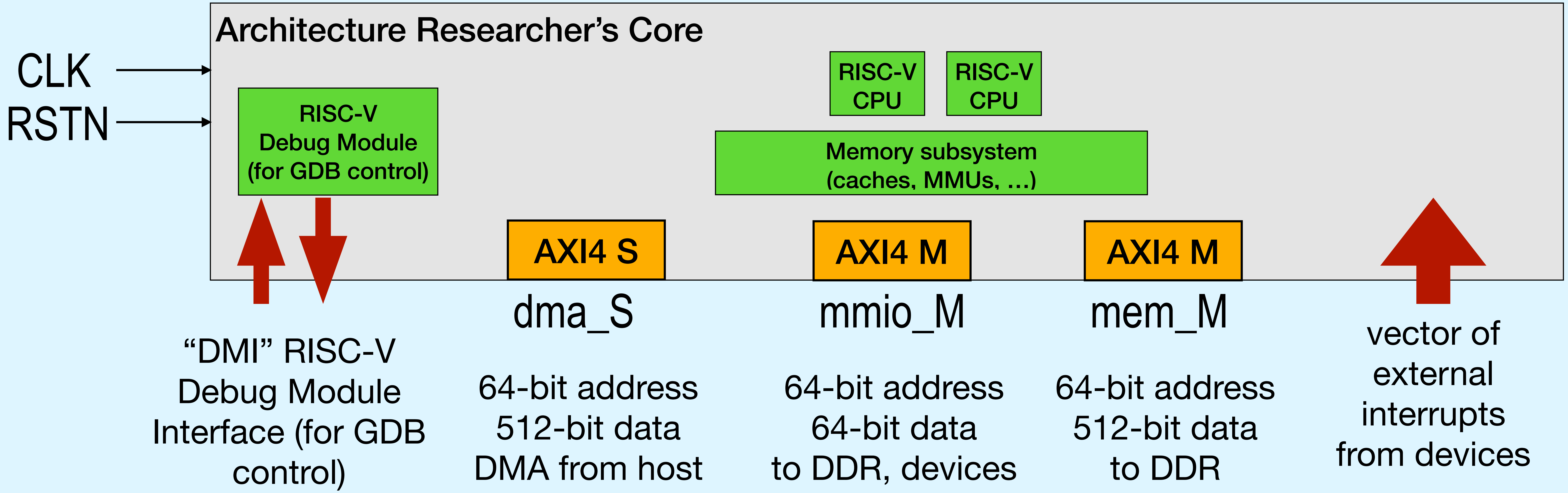
- In some cores, mem_M transfers entire cache lines at the back-end of an L2 cache
- In some cores, dma_S connects to a cache-coherent port in the memory subsystem

Plug in your own RISC-V core

The Catamaran/ARIFIC Core Interface (RTL)

DEMO: View actual interface RTL in tutorial repository: Catamaran_Core_Interface/mkAWSteria_Core_EMPTY.v

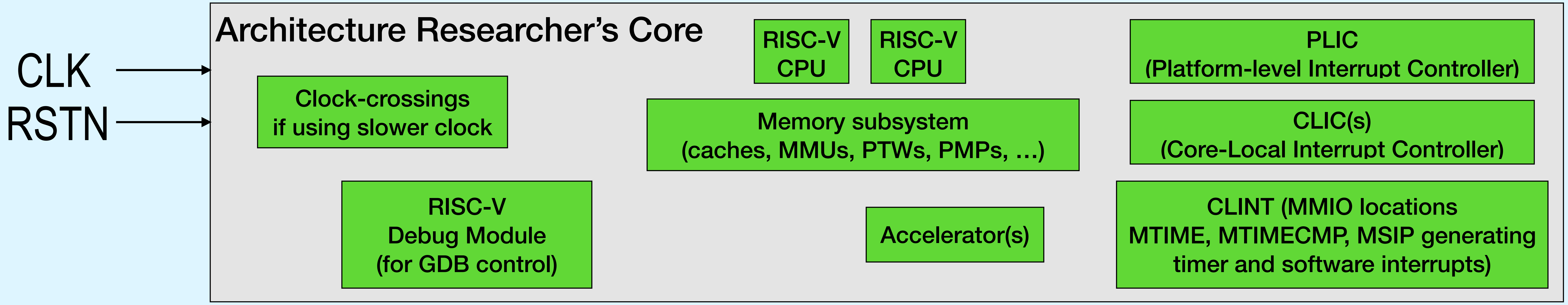
More interfaces



Plug in your own RISC-V core

The Catamaran/ARIFIC Core Interface (RTL)

Typical modules found in more complete and complex cores



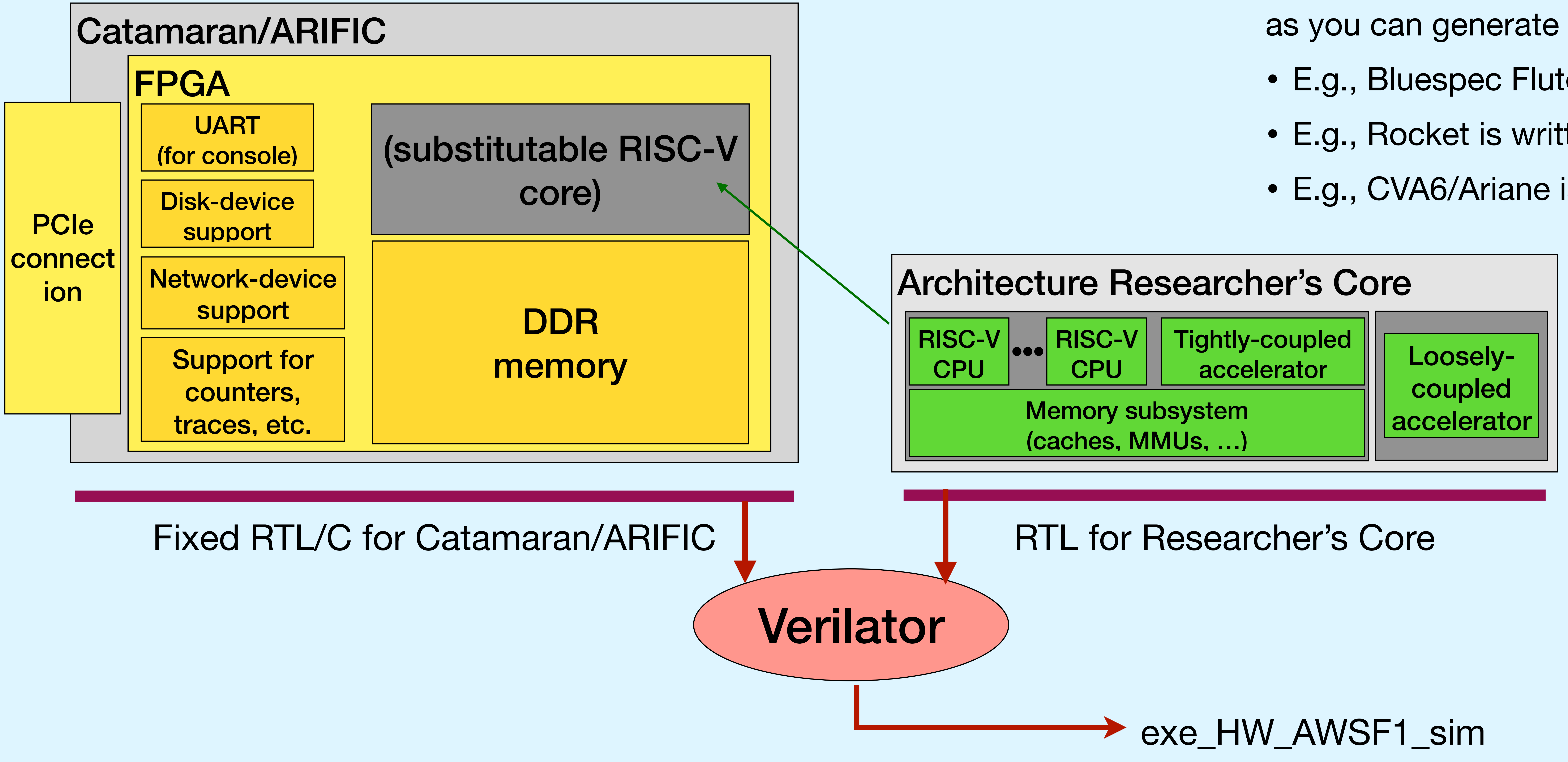
- Catamaran provides 6 clocks at various steps from 250MHz down to about 15 MHz
- All these components are available open-source from many projects; architecture researcher can focus on the subject module(s) of interest.

Plug in your own RISC-V Core

- The Catamaran/ARIFIC Core Interface (RTL)
- *DEMO: Build and run a whole-system simulation executable*
- DEMO: Build an FPGA bitfile for Amazon AWS

Plug in your own RISC-V core

Build a whole-system simulation executable



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

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

Plug in your own RISC-V core

DEMO: Build a whole-system simulation executable

This tutorial repository contains
- Fixed RTL for Catamaran/ARIFIC
- 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

```
Build_HW/
    Catamaran_C/
    Catamaran_RTL/
    Core_CVA6_Wrapper_RTL/
    bsc_lib_RTL/
```

https://github.com/openhwgroup/cva6.git

Fixed RTL/C for Catamaran/ARIFIC

RTL from OpenHardware Group CVA6 (Ariane) RISC-V Core repository

Verilator

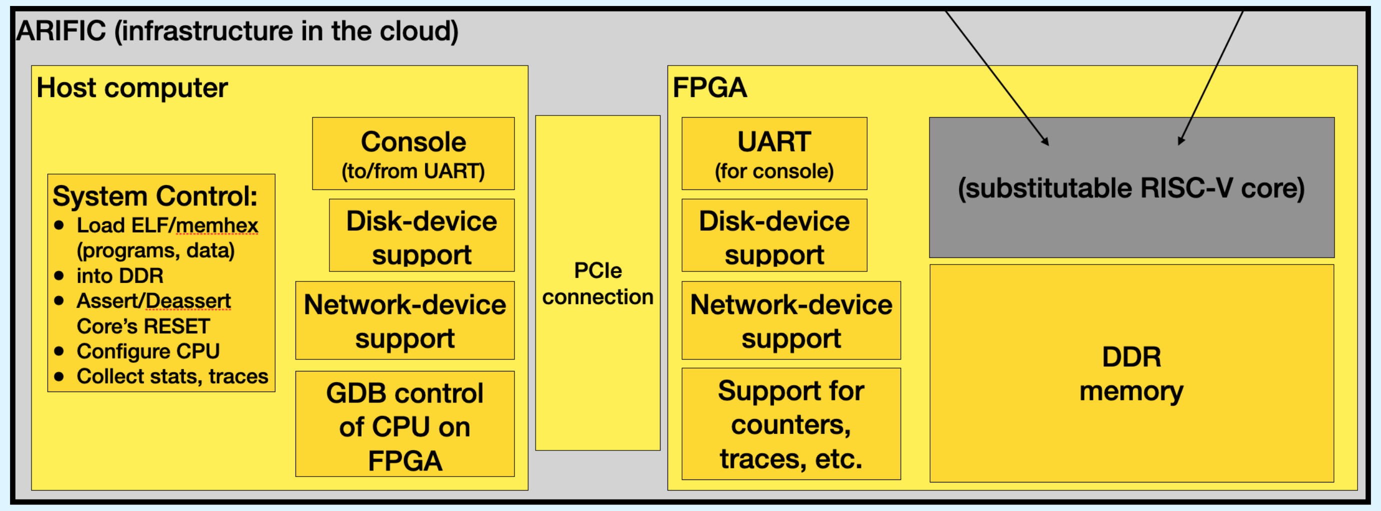
$ cd Build_HW; make exe_HW_AWSF1_sim

Note: does not need Amazon AWS;
can be built on your laptop/desktop.

exe_HW_AWSF1_sim

Plug in your own RISC-V core

DEMO: Run a whole-system simulation executable



exe_Host_ISA_Tester_AWSF1_sim

Catamaran_simulation_Host

```
[$ ./exe_Host_ISA_Tester_AWSF1_sim  ../ISA_Tests/isa/elfs  rv64ui-p-add
Directory for ELF files: ../ISA_Tests/isa/elfs
File-selection pattern:  rv64ui-p-add
tcp_client_open: connecting to '127.0.0.1' port 30000
tcp_client_open: connected
INFO: DRM check
INFO: DRM: telling HW-side to disallow execution
INFO: DRM: telling HW-side to allow execution
INFO: DRM: Waiting for HW-side to confirm allow-execution
    ... confirmed after 1 polls
--------------------------------
Asserting reset (stops CPU)
--------------------------------
Loading ELF file into RISC-V memory:
  ../ISA_Tests/isa/elfs/rv64ui-p-addi
  ELF load complete
  Transfer rate: 3351648 bytes/sec (1220 bytes in 364000 nsecs)
  Readback check
  Readback check complete
Deasserting reset (CPU starts running)
Asserting reset (stops CPU)
    Test PASS
So far: 1 test files, 1 PASS, 0 FAIL, 0 TIMEOUT
--------------------------------
```

Instead of PCIe, connect with TCP/IP

Does not need Amazon AWS;
can run on your laptop/desktop

exe_HW_AWSF1_sim

Catamaran_simulation_HW

```
--13:05:30--Airedale: ~/Git/Tutorial_at_HPCA-29/Build_HW
[$ ./exe_HW_AWSF1_sim
================================================================
Bluespec Catamaran simulation v2.2
Copyright (c) 2020-2023 Bluespec, Inc. All Rights Reserved.
================================================================
INFO: Listening for connection from host-side on TCP port 30000
INFO: Accepted connection from host-side on TCP port 30000
Host_Control_Status: Assert Core Reset
Host_Control_Status: Deassert Core Reset
Host_Control_Status: Assert Core Reset
Host_Control_Status: Deassert Core Reset
Host_Control_Status: Assert Core Reset
Host_Control_Status: Deassert Core Reset
Host_Control_Status: Assert Core Reset
Host_Control_Status: Deassert Core Reset
Host_Control_Status: Assert Core Reset
Connection closed by remote host (in c_host_recv2())
--13:10:44--Airedale: ~/Git/Tutorial_at_HPCA-29/Build_HW
$ 
```

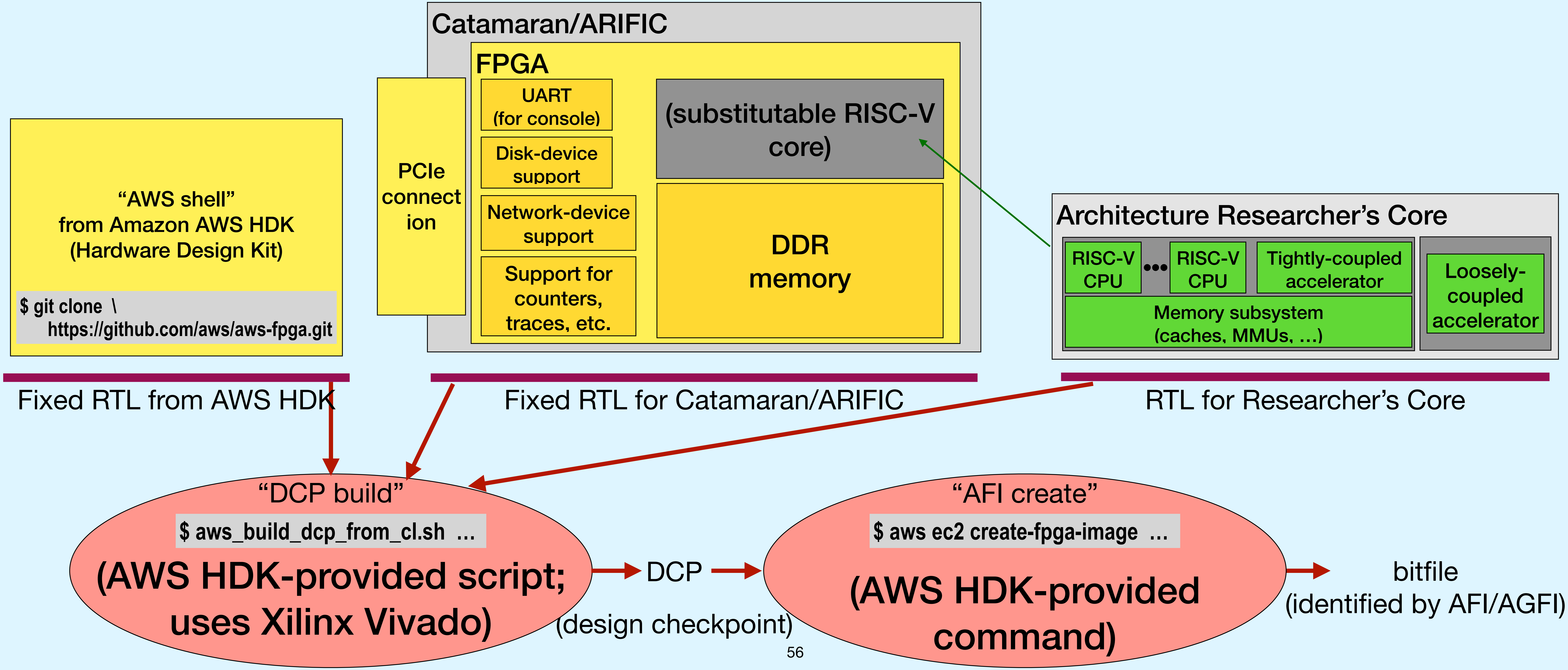
- Simulation is, of course, much slower than running on FPGA Simulation (Linux boot can take a day or more)
- But simulation can print RTL $displays, dump VCD waveforms, etc.
- So: simulation is good for running/debugging with small RISC-V programs

Plug in your own RISC-V Core

- The Catamaran/ARIFIC Core Interface (RTL)
- DEMO: Build and run a whole-system simulation executable
- *DEMO: Build an FPGA bitfile for Amazon AWS*

Plug in your own RISC-V core

Building an FPGA bitfile for Amazon AWS



Plug in your own RISC-V core

DEMO: Building an FPGA bitfile for Amazon AWS

General Caveat

- The flow described here and the files currently in the tutorial repository are somewhat specialized for building CVA6/Ariane RV64GC_MSU_Sv39 as the RISC-V Core in Catamaran.
- For a different core (and therefore different core RTL)
  - Various Vivado Tcl scripts have to be modified accordingly
  - Other adjustments may be needed to select a different clock speed
  - Other adjustments may be needed to incorporate a RISC-V Debug Module, which must sit outside the “reset domain” of the core
- We expect to simplify some of these adjustments in the weeks/months after the tutorial

Plug in your own RISC-V core

DEMO: Building an FPGA bitfile for Amazon AWS

“DCP build” step (this example is for the CVA6/Ariane core)

- Prepare a directory that is ready for the AWS HDK flow:

```
$ cd  Build_HW
$ make  for_aws
```

  - Creates a directory “AWS_CL_Catamaran/” which is initialized from “AWS_CL_Catamaran_wo_Core.tar.gz”, i.e., fixed RTL for Catamaran
  - Then, copies the core’s RTL from your clone of https://github.com/openhwgroup/cva6.git (in the Makefile, “CVA6_REPO_DIR=…” should be edited to point at this clone)
  - Finally, tars up this directory into “AWS_CL_Catamaran_for_AWS_HDK.tar.gz”

- Copy this tar file to your “build AMI”, where you have Vivado tools and the AWS aws-fpga HDK+SDK installed, and untar it.
- Launch the DCP build:

```
$ cd  AWS_CL_Catamaran/build/scripts
$ ./aws_build_dcp_from_cl.sh   -ignore_memory_requirement   -clock_recipe_a  A1
```

  - This will launch a background process to perform Vivado synthesis resulting in a DCP (Design Checkpoint)
  - The “clock_recipe” argument here sets the core’s clock frequency to 125MHz
  - The Vivado log is saved in a file with a timestamped filename, like this: “23_02_21-012910.vivado.log”. You should monitor this file to check that Vivado has not exited due to some error.
  - This step takes ~ 2.5 hours for CVA6 RV64 GC SMU Sv39.

- When Vivado has finished, you should examine some files to check that it succeeded:
  - “build/scripts/<*timestamp*>.vivado.log” for any errors encountered by Vivado
  - “build/reports/<*timestamp*>.timing_summary_route_design.rpt” to check that synthesis met the timing target

See AWS HDK documentation for more info on clock recipes, email-notification of synthesis completion, etc.

Plug in your own RISC-V core

DEMO: Building an FPGA bitfile for Amazon AWS

“AFI create” step: Launching the AFI-build

- A successful “DCP build” step produces a design checkpoint file: “build/checkpoints/to_aws/23_02_21-012910.Developer_CL.tar”
- Create an S3 bucket (if you have not already done so before):

```
$ aws s3 mb  s3://rsnbucket1/Catamaran/
```

  - “S3” is Amazon AWS’ name for their “storage in the cloud” service
  - “bucket” is Amazon AWS’ name for “top-level directory”

- Upload your design checkpoint file to your S3 bucket:

```
$ aws s3 cp  <path>/<timestamp>.Developer_CL.tar  s3://rsnbucket1/Catamaran/
```

- Launch the AFI build

See AWS HDK documentation for more info

```
$ aws ec2 create-fpga-image  --region us-east-1 \
  —name “Catamaran_CVA6_RV64GC_MSU_Sv39_Boot_ROM_125MHz"  \
  —description "Catamaran CVA6 RV64GC MSU Sv39 Boot ROM_125MHz”  \
  —input-storage-location Bucket=rsnbucket1,Key=Catamaran/<timestamp>.Developer_CL.tar  \
  —logs-storage-location Bucket=rsnbucket1,Key=Catamaran
```

- This will immediately print out the AFI and AGFI allocated for this new bitfile:

```
{
    "FpgaImageId": "afi-0a3fa15056f3de4d7",
    "FpgaImageGlobalId": "agfi-0ab786417c04b7031"
}
```

IMPORTANT: please note these down immediately! You have no other way to refer to your bitfile other than these IDs!

Plug in your own RISC-V core

DEMO: Building an FPGA bitfile for Amazon AWS

“AFI create” step: Checking on your AFI build

- The following command prints info about an AFI:

```
$ aws ec2 describe-fpga-images --fpga-image-ids  "afi-0a3fa15056f3de4d7"
```

- This will print out something like this:


```
{
  "FpgaImages": [
    {
      "UpdateTime": "2023-02-21T16:28:22.000Z",
      "Name": "Catamaran_CVA6_RV64GC_MSU_Sv39_Boot_ROM_125MHz",
      "Tags": [],
      "PciId": {
        "SubsystemVendorId": "0xfedc",
        "VendorId": "0x1d0f",
        "DeviceId": "0xf001",
        "SubsystemId": "0x1d51"
      },
      "DataRetentionSupport": true,
      "FpgaImageGlobalId": "agfi-0ab786417c04b7031",
      "Public": false,
      "State": {
        "Code": "available"
      },
      "ShellVersion": "0x04261818",
      "OwnerId": "071524437452",
      "FpgaImageId": "afi-0a3fa15056f3de4d7",
      "CreateTime": "2023-02-21T15:32:09.000Z",
      "Description": "Catamaran CVA6 RV64GC MSU Sv39 Boot ROM_125MHz"
    }
  ]
}
```


Finish time of AFI build

“pending” at start of AFI build;
“available” at finish of AFI build

Start time of AFI build

AFI build time ~ 55 minutes. Seems independent of complexity of user’s design (just stitches together user’s DCP with AWS boilerplate)

The AFI is now ready for use (load into FPGA, …)

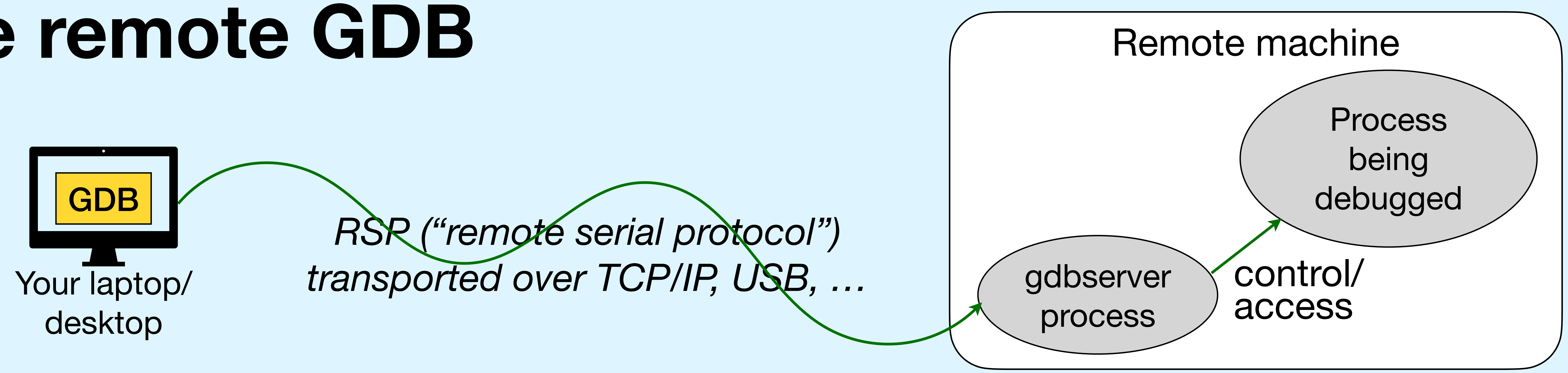
See AWS HDK documentation for more info

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

GDB control of RISC-V CPU on FPGA

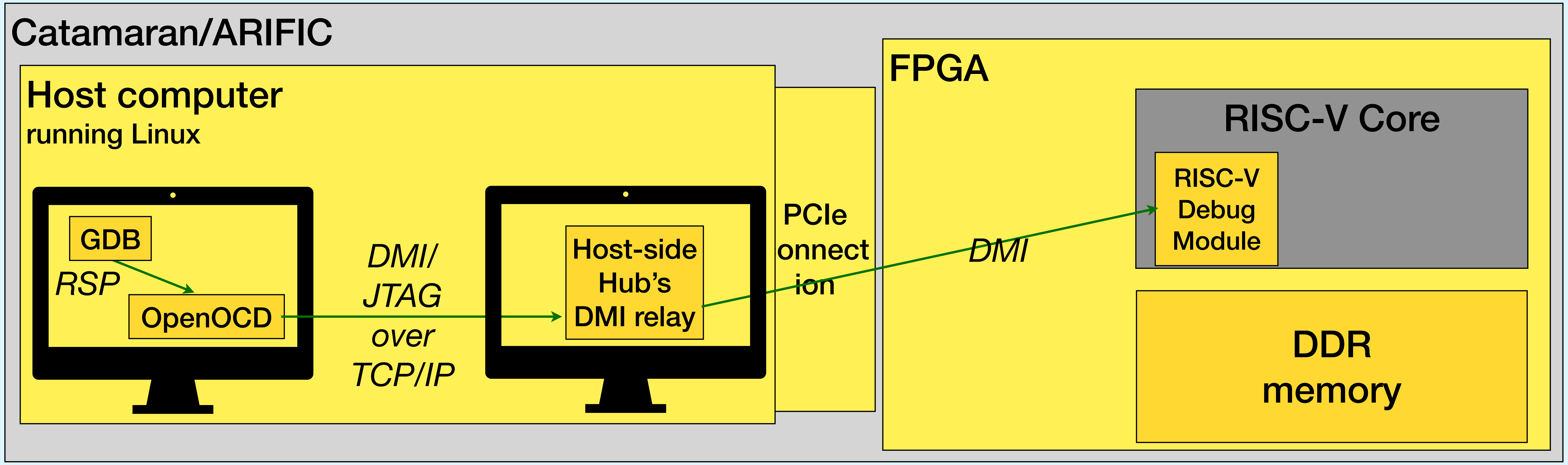
Traditional vs. hardware remote GDB

Traditional “remote debugging” is done entirely with software:



This assumes that the hardware (remote machine) is reliable (e.g., to run gdbserver); this may not be true for the architecture researcher.

On Catamaran:



GDB control of RISC-V CPU on FPGA

Connecting GDB

- When you start the Host-side Hub, provide a TCP/IP port number for GDB to connect:

```
$ cd  ~/Git/Tutorial_at_HPCA-29/Host_side_exes
$ ./exe_Host_AWSF1  —debugport  5555
```

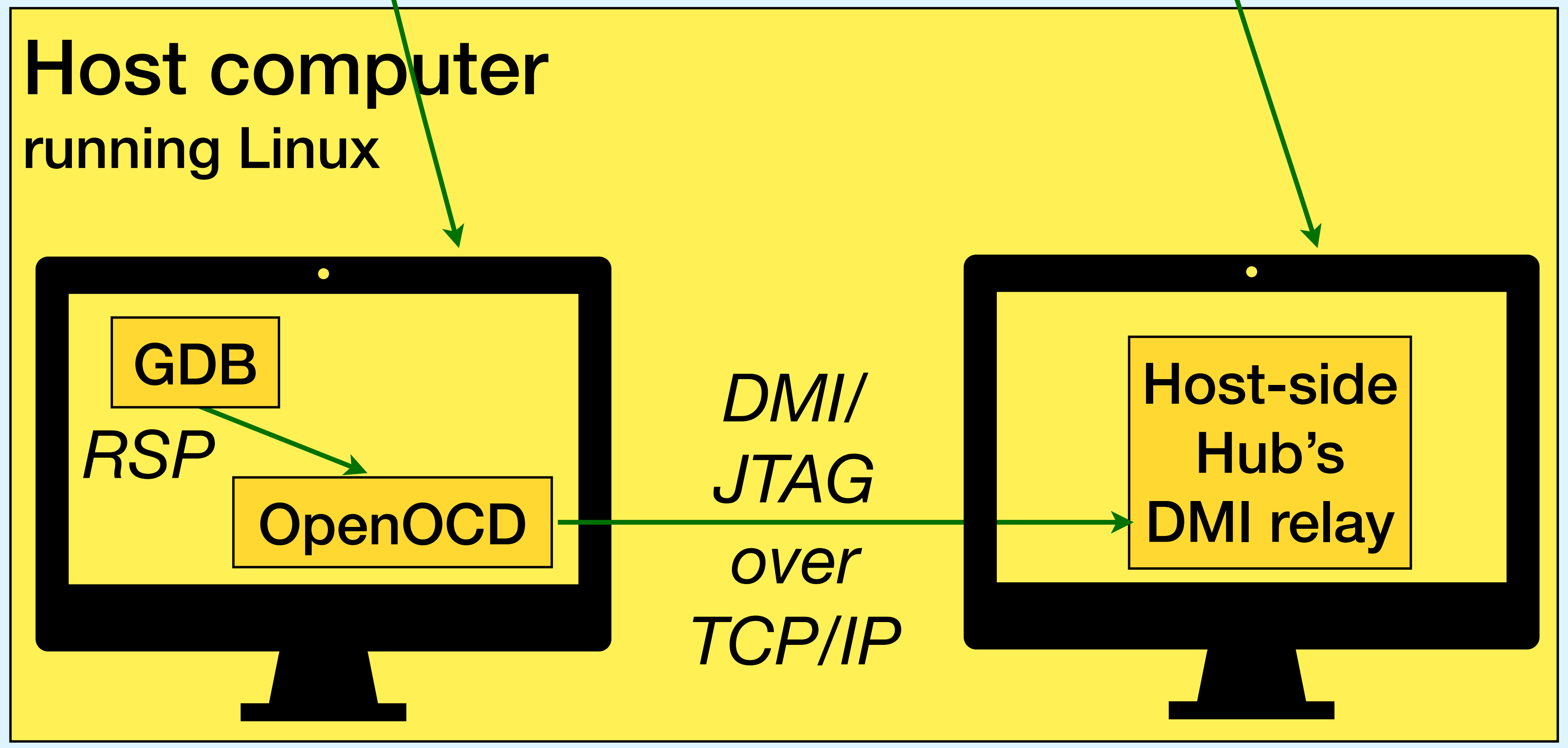
- In another terminal window, start GDB with the provided script, which will start GDB and OpenOCD and make the connections, and pause at the usual GDB prompt:

```
$ cd ~/Git/Tutorial_at_HPCA-29/GDB_OpenOCD
$ riscv64-unknown-elf-gdb  —silent  —command init_64.gdb
…
(gdb)
```

Please peruse the supplied gdb and OpenOCD scripts for more details of how the connections are established, etc.

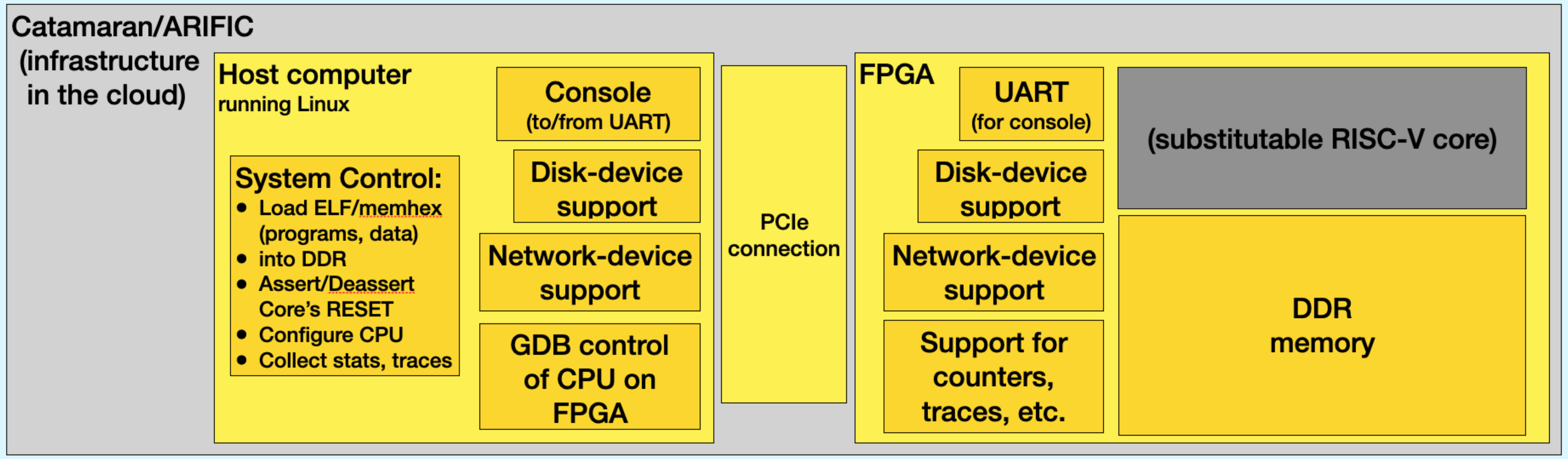
- At the (gdb) prompt, use GDB commands as usual:
  load ELF,
  start, stop, continue, step, step, break,
  read/write registers and CSRs and memory, …

Hint: Loading an ELF from GDB can be slow; you can still use the "loadelf" command at the Host-side Hub prompt to load an ELF, much faster.



Summary and Conclusion

Summary of tutorial



- Catamaran/ARIFIC provides the large and complex infrastructure that frees the Architecture Researcher to focus on their creative work: research into the CPU microarchitectures, memory systems, accelerators, etc.
- In this tutorial we showed the capabilities immediately enabled by Catmaran/ARIFIC:
    - 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

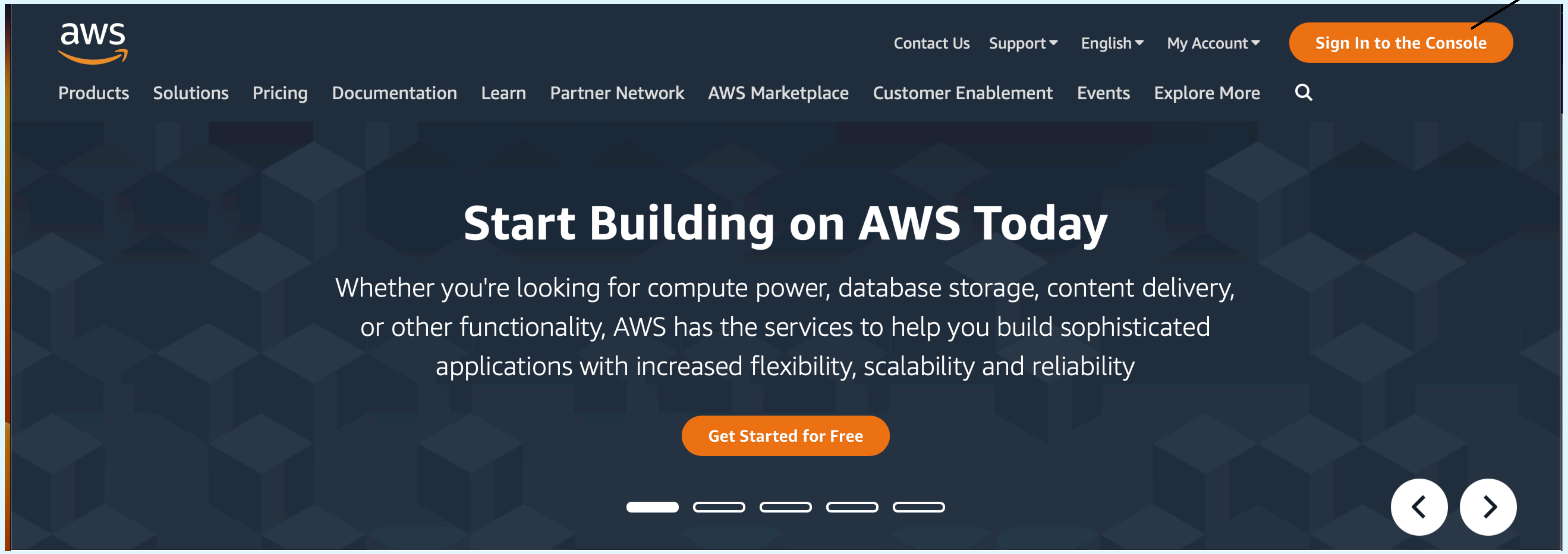
Appendix: Reference Material

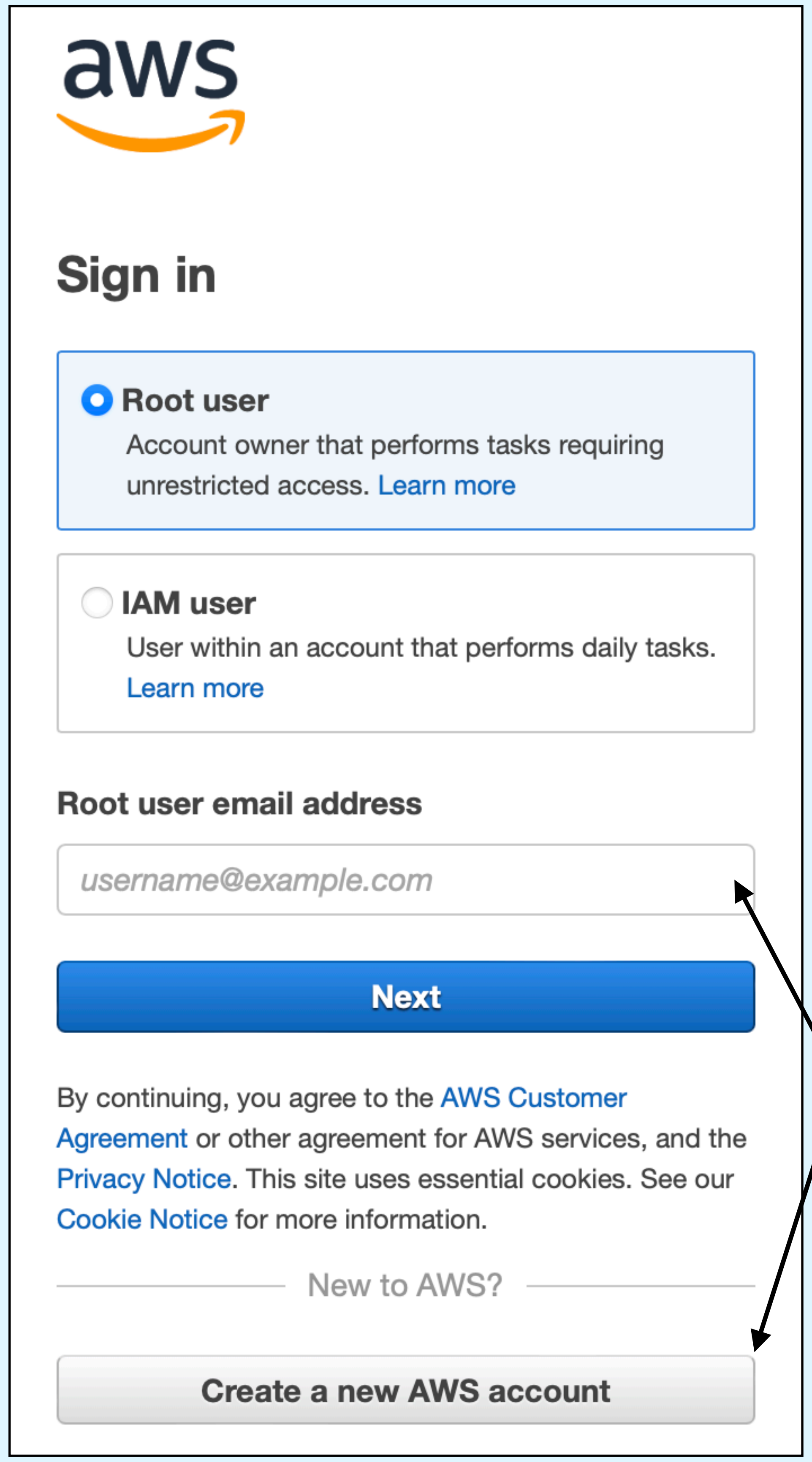
- *Open an account on Amazon AWS*
- Create an AMI (or two) for Catamaran/ARIFIC
- Install Catamaran software and scripts
- Address map
- Install the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)
- Install aws-fpga HDK and SDK
- Install the “aws” CLI (command-line interface)
- Install XDMA driver for Host-FPGA PCIe communication

*Disclaimer: this **not** official AWS documentation, and is only based on this author's observations.*

Open an account on Amazon AWS

- 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

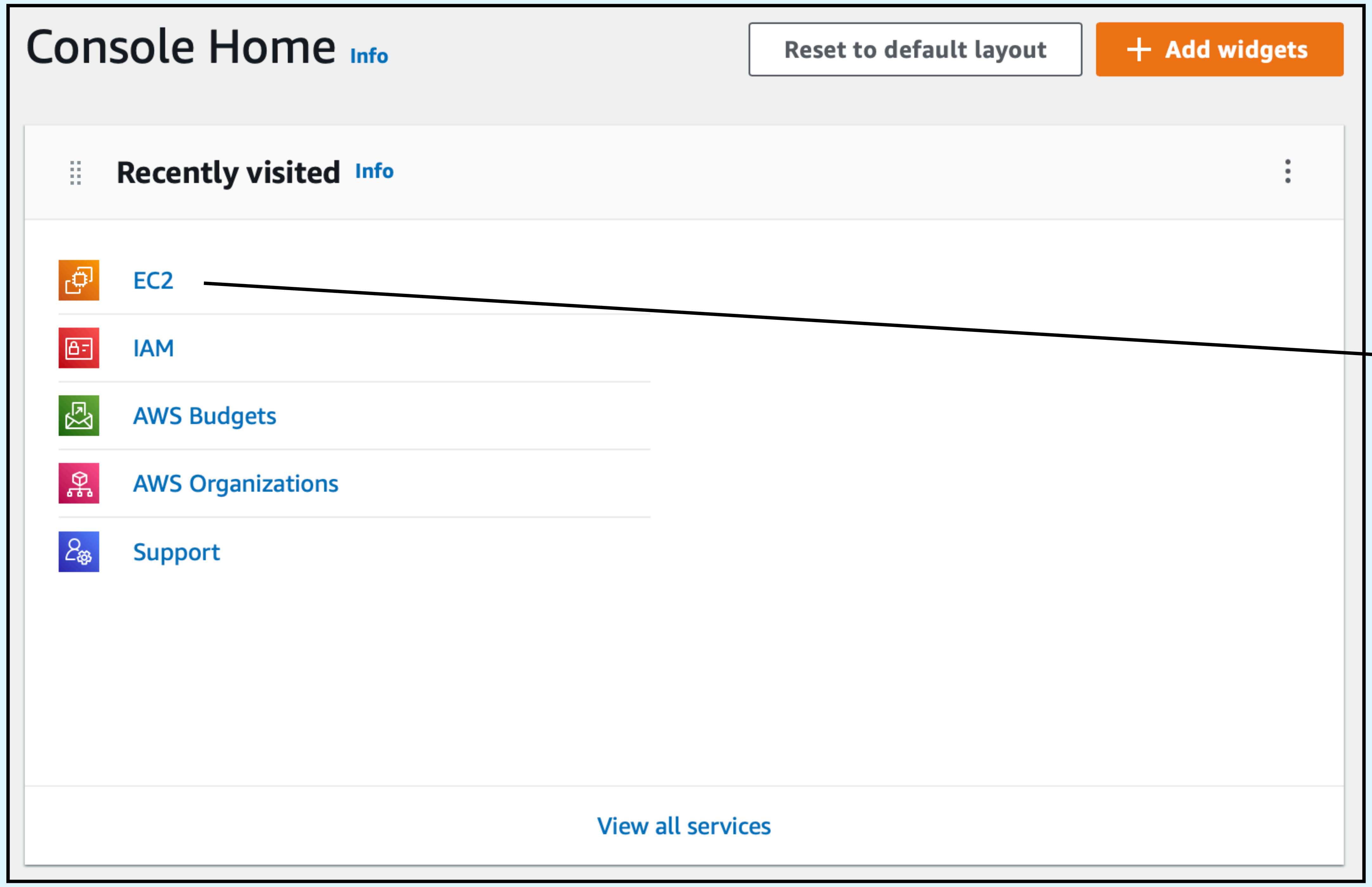
Appendix: Reference Material

- Open an account on Amazon AWS
- *Create an AMI (or two) for Catamaran/ARIFIC*
- Install Catamaran software and scripts
- Address map
- Install the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)
- Install aws-fpga HDK and SDK
- Install the “aws” CLI (command-line interface)
- Install XDMA driver for Host-FPGA PCIe communication

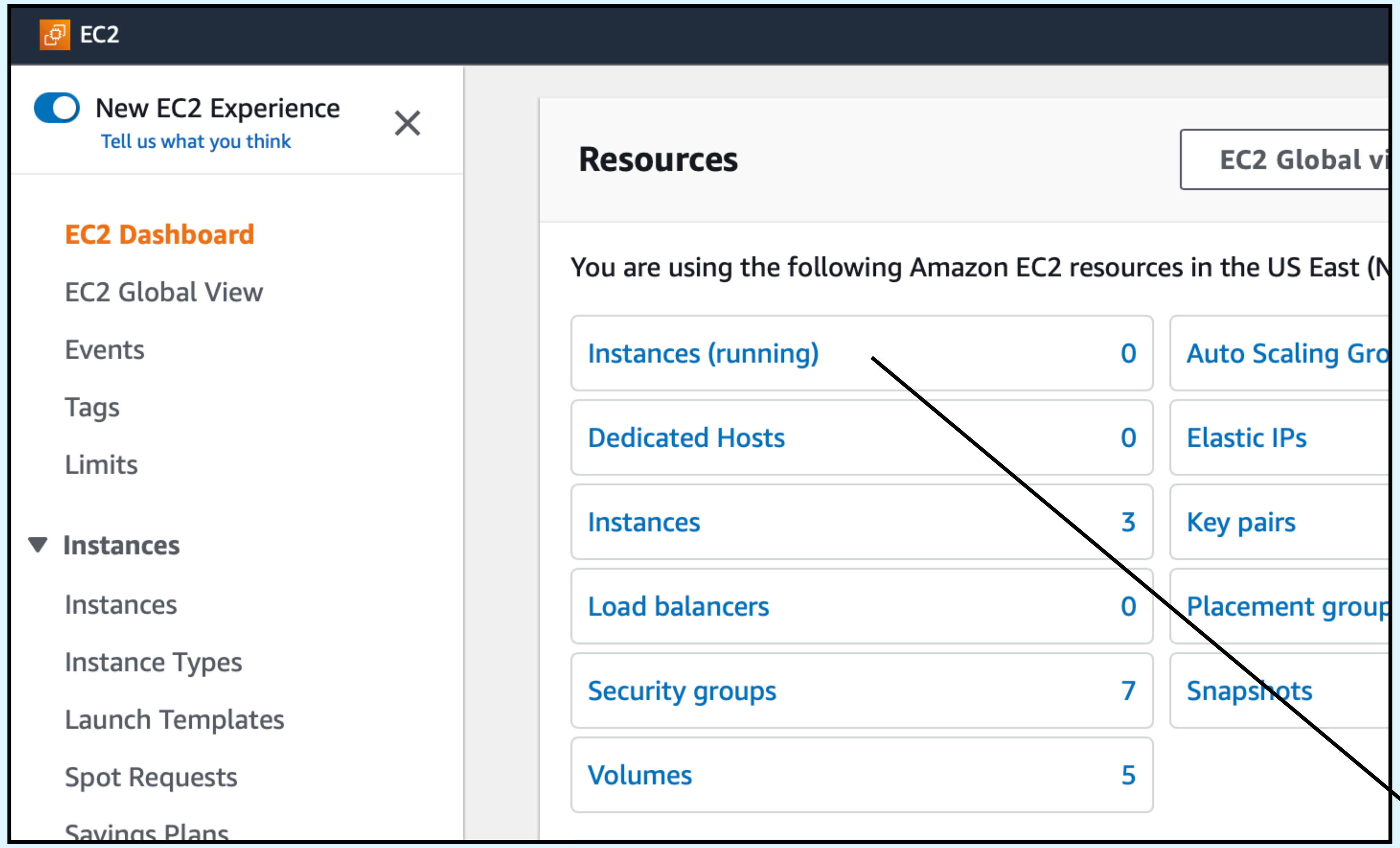
Disclaimer: this **not** official AWS documentation, and is only based on this author's observations.

Create an AMI (or two) for Catamaran/ARIFIC

Navigate to your EC2 Instances dashboard



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



●●●

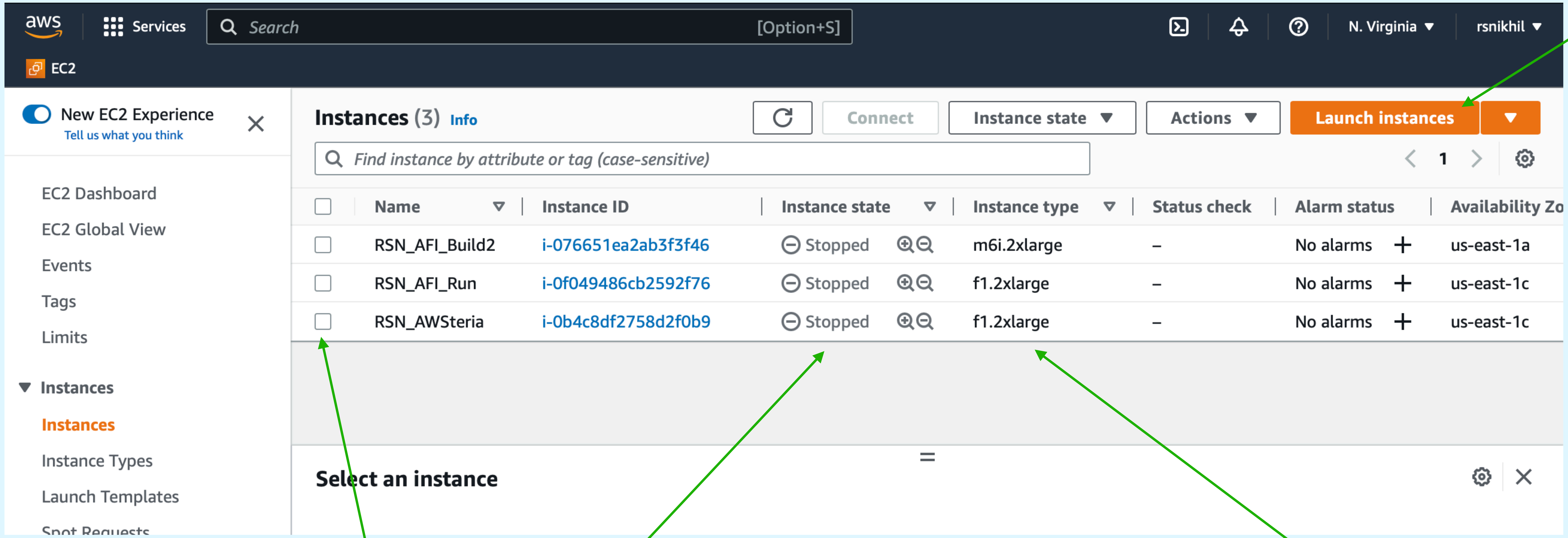
Then, move on to the "Instances" dashboard
("instances" = AWS terminology for your virtual machines)

Disclaimer: *this **not** official AWS documentation, and is only based on this author's observations.*

Create an AMI (or two) for Catamaran/ARIFIC

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.

*Disclaimer: this **not** official AWS documentation, and is only based on this author's observations.*

Create an AMI (or two) for Catamaran/ARIFIC

Create one or two AMIs?

The "Instance type" determines, for your AMI:

- Host CPU type (x86, ARM, …)
- Sizing (# of virtual cores, memory, …)

Note: "f1.*" types are the *only* ones with FPGAs attached.
Note: AWS charges vary with instance type, and "f1" instances are typically more expensive.

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. (all 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

Disclaimer: *this **not** official AWS documentation, and is only based on this author's observations.*

"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)

Disclaimer: this **not** official AWS documentation, and is only based on this author's observations.

"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

- Open an account on Amazon AWS
- Create an AMI (or two) for Catamaran/ARIFIC
- *Install Catamaran software and scripts*
- Address map
- Install the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)
- Install aws-fpga HDK and SDK
- Install the “aws” CLI (command-line interface)
- Install XDMA driver for Host-FPGA PCIe communication

Install Catamaran Software and Scripts

What are Catamaran software and scripts?

Executable	Description
exe_Host_AWSF1	Full host-side for running arbitrary ELF on Catamaran on FPGA supporting connections to UART console, GDB/OpenOCD, with network and block-storage device support, etc.
exe_Host_AWSF1_sim	Version of above for running with a hardware simulation instead of FPGA
exe_Host_ISA_Tester_AWSF1	Host-side to run a series of ISA tests on Catamaran on FPGA
exe_Host_ISA_Tester_AWSF1_sim	Version of above for running with a hardware simulation instead of FPGA
netinit.sh	Host-side script to prepare TUN/TAP network device before booting Linux on FPGA
Octopus1o_tuntap.elf	ELF file containing OpenSBI bootloader + minimal Linux

Install Catamaran Software and Scripts

Installation

Bluespec, Inc. has made these available using the usual “apt” package-installation mechanism (Debian, Ubuntu).

- This command will download Bluespec, Inc.’s public key for authenticating the “apt” step:

```
$ curl -sS https://s3.wasabisys.com/bluespec/downloads/apt/bluespec.asc | sudo tee /etc/apt/trusted.gpg.d/bluespec.asc
```

- This command will inform your “apt” installer about where to go for the download:

```
$ curl -sS https://s3.wasabisys.com/bluespec/downloads/apt/ubuntu/bluespec-jammy.sources |  \
      sudo tee /etc/apt/sources.list.d/bluespec-jammy.sources
```

- Finally, this command will download and install the toolchains:

```
$ sudo apt install  bluespec-catamaran-common
```

Install Catamaran Software and Scripts

Where the software and scripts are installed

Installing the Catamaran Software and Scripts, per the previous slide, results in:

```
$ tree -d /opt/bluespec/catamaran-common-1.20230222.0.1/
/opt/bluespec/catamaran-common-1.20230222.0.1/
├── Host
│   └── bin
├── Templates
│   ├── Prebuilt
│   ├── cat
│   │   └── lib
│   ├── hello
│   │   └── lib
│   └── hellocpp
│       └── lib
├── UART_Console
│   ├── bin
│   └── src
├── bin
├── isa-tests
│   └── isa
│       └── elfs
└── linux
    └── kernel
```

exe_Host_AWSF1
exe_Host_ISA_Tester_AWSF1
UART_Console

Octopus1o_tuntap.elf
(OpenSBI boot loader + Linux kernel for RISC-V)

Appendix: Reference Material

- Open an account on Amazon AWS
- Create an AMI (or two) for Catamaran/ARIFIC
- Install Catamaran software and scripts
- *Address map*
- Install the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)
- Install aws-fpga HDK and SDK
- Install the “aws” CLI (command-line interface)
- Install XDMA driver for Host-FPGA PCIe communication

Install Catamaran Software and Scripts

Address map for Catamaran

If you create your own software to run on the RISC-V core in Catamaran, or rebuild your own Linux, you will need some or all of the following address information (typically used in your “ld” linker script.

Device	Start address (hex)	Size (hex)	MMIO/Mem	Location in system
Boot ROM	0001_0000	0001_0000	MMIO	Core
CLINT	0200_0000	0001_0000	MMIO	Core
PLIC	0C00_0000	0400_0000	MMIO	Core
Host access	4000_0000	0000_4000	MMIO	Catamaran system
UART	6010_0000	0000_1000	MMIO	Catamaran system
MSI	6020_0000	0000_0004	MMIO	Catamaran system
SRAM	7000_0000	0040_0000	MMIO	Catamaran system
DDR	8000_0000	8000_0000	Mem	Catamaran system

PC reset value: 0x_1_0000 (i.e., start of Boot ROM)

Install Catamaran Software and Scripts

Address map for Catamaran: core device details

These devices are in the substitutable Catamaran Core, and are "the same" in the provided cores (Rocket, CVA6, Flute). Other cores may omit them, have different functionality, or different address maps.

Boot ROM: Typically contains the "device tree" for Linux and some initial instructions that call the Linux boot loader (e.g., OpenSBI) passing it the address of the device tree.

CLINT: Contains standard memory-mapped locations (see RISC-V Privileged ISA Specification for details).

- MSIP: offset 0, size 8 bytes: inter-hart software interrupts
- MTIME: offset 0xBFF8, size 8 bytes: real-time counter
- MTIMECMP: offset 0x4000, size 8 bytes: real-time compare, generating a timer interrupt

PLIC: Platform-level Interrupt Controller

- Please see "SiFive U54-MC Core Complex Manual v1p0", https://static.dev.sifive.com/U54-MC-RVCoreIP.pdf for functional details and internal address-map details.

Install Catamaran Software and Scripts

Address map for Catamaran: Catamaran system device details

These devices are in the fixed Catamaran System.
Addresses can be changed by regenerating the Catamaran System RTL.

Host Access: loads/stores to these locations are handled remotely by the x86 host. We use these for:

- “Virtio” device locations (networking device, block device, entropy device)
- “tohost” location used by ISA tests

UART: this is hardware that emulates an NS16550 UART. Please see documentation widely available on the Web for internal register addresses and functionality.

MSI: “Message Sequenced Interrupts”: a write to this location from the x86 host delivers an external interrupt to the PLIC and thence to the RISC-V CPU. we use these typically for interrupts from “Virtio devices” emulated on the host-side.

SRAM: Just a region of SRAM in MMIO (uncached) space.

DDR: This is main memory for the RISC-V CPU, which can be cached. Has high-speed path (512-bit AXI4 with bursts) to/from the CPU.

Appendix: Reference Material

- Open an account on Amazon AWS
- Create an AMI (or two) for Catamaran/ARIFIC
- Install Catamaran software and scripts
- Address map
- *Install the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)*
- Install aws-fpga HDK and SDK
- Install the “aws” CLI (command-line interface)
- Install XDMA driver for Host-FPGA PCIe communication

Install the RISC-V Gnu Toolchain

What is the “RISC-V Gun Toolchain”?

- The familiar tools: *gcc/g++* (compilers), *as* (assembler), *ld* (linker), *gdb* (debugger), *objdump* (disassembler), …
- 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 (ELF binaries).
  - This could be your “run” AMI on Amazon AWS,
  - or any Ubuntu machine.

In the next few slides we describe two options:

- Simply download and install pre-built toolchains (pre-built by Bluespec, Inc.)
  - This is good enough for use for Catamaran
- Some tips on how you can build your own toolchain from sources
  - If you want to do something more sophisticate, customized, …

Install the RISC-V Gnu Toolchain

Download and install pre-built toolchains

Bluespec, Inc. has pre-built certain configurations of the RISC-V Gnu Toolchain,

and made them available using the usual “apt” package-installation mechanism (Debian, Ubuntu).

- This command will download Bluespec, Inc.’s public key for authenticating the “apt” step:

```
$ curl -sS https://s3.wasabisys.com/bluespec/downloads/apt/bluespec.asc | sudo tee /etc/apt/trusted.gpg.d/bluespec.asc
```

- This command will inform your “apt” installer about where to go for the download:

```
$ curl -sS https://s3.wasabisys.com/bluespec/downloads/apt/ubuntu/bluespec-jammy.sources |  \
      sudo tee /etc/apt/sources.list.d/bluespec-jammy.sources
$ sudo apt update
```

- Finally, this command will download and install the toolchains:

```
$ sudo apt install  bluespec-riscvtoolchain-medany
```

Install the RISC-V Gnu Toolchain

Where the pre-built toolchains are installed

Installing the pre-built RISC-V Gnu Toolchains, per the previous slide, results in:

```
$ tree -d -L 3 /opt/bluespec/
/opt/bluespec/
├── riscvtoolchain-medany -> riscvtoolchain-medany-1.202211.1
└── riscvtoolchain-medany-1.202211.1
    ├── bin
    ├── openocd
    │   ├── bin
    │   ├── share
    │   └── src
    └── toolchains
        ├── baremetal-medany
        ├── linux-multilib-medany
        └── src
```

The “`bin/`” directory contains the Gnu tools: gcc and g++ compilers, gdb, ar, as, ld, nm, objdump, …

including the compiler for “C on bare-metal”:

```
/opt/bluespec/riscvtoolchain-medany/bin/riscv64-unknown-elf-gcc
```

and the compiler for C-under-Linux:

```
/opt/bluespec/riscvtoolchain-medany/bin/riscv64-unknown-linux-gnu-gcc
```

Install the RISC-V Gnu Toolchain

Building and installing the toolchain from scratch

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*

Disclaimer: this **not** official RISC-V or GNU documentation, and is only based on this author's observations.

Install the RISC-V Gnu Toolchain

Using the toolchain

Before using the tools in the toolchain, please export the following environment variables:

```
$ export RISCV=<path_to_installation_dir>
#                 e.g., /opt/bluespec/riscvtoolchain-medany

$ export PATH=${RISCV}/bin:${PATH}
```

Appendix: Reference Material

- Open an account on Amazon AWS
- Create an AMI (or two) for Catamaran/ARIFIC
- Install Catamaran software and scripts
- Address map
- Install the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)
- *Install aws-fpga HDK and SDK*
- Install the “aws” CLI (command-line interface)
- Install XDMA driver for Host-FPGA PCIe communication

*Disclaimer: this **not** official AWS documentation, and is only based on this author's observations.*

Install 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

- Open an account on Amazon AWS
- Create an AMI (or two) for Catamaran/ARIFIC
- Install Catamaran software and scripts
- Address map
- Install the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)
- Install aws-fpga HDK and SDK
- *Install the "aws" CLI (command-line interface)*
- Install XDMA driver for Host-FPGA PCIe communication

*Disclaimer: this **not** official AWS documentation, and is only based on this author's observations.*

Install the "aws" CLI (command-line interface)

The "aws" CLI is a software tool provided free by AWS.

- Is used for all kinds of AWS activities, including creating/listing/populating S3 buckets (storage in the cloud), describing AFIs (FPGA images), … For example:

```
$ aws  ec2  describe-fpga-images  --fpga-image-ids  $(AFI_ID)
```

When logged in to your AMI, install the CLI using the package manager for your AMI's OS:

```
$ sudo apt-get install awscli        # Ubuntu or Debian
$ sudo yum install awscli            # CentOS, ...
```

Detailed information about AWS CLI:

https://aws.amazon.com/cli/

Appendix: Reference Material

- Open an account on Amazon AWS
- Create an AMI (or two) for Catamaran/ARIFIC
- Install Catamaran software and scripts
- Address map
- Install the RISC-V Gnu Toolchain (gcc, as, ld, gdb, …)
- Install aws-fpga HDK and SDK
- Install the “aws” CLI (command-line interface)
- *Install XDMA driver for Host-FPGA PCIe communication*

Install 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