Running LHCb Reconstruction on ARM

Ben Couturier, Marco Clemencic,
Niko Neufeld, **Vijay Kartik**

PH-LBC

Concurrency Forum

24/04/2013

Outline

- ARM – Initial comparison of systems
- Brief current status
- LHCb Event Reconstruction on ARM
- Comparisons with x86_64
- Future work

Is it worth it?

	In-rack size	Frequency (GHz)	Cores	HEP-SPEC	Coremark	Power (W)
Intel X5650	2U	2.67	96 Hyperthreads	900	7715	1400
ARM Cortex-A8	2U	~1	192	520(58%)	3165(40%)	**300**
ARM Cortex-A9	Dev. kit	~1.3	4	-	3425(44%) **gcc-47/45** 2985(38%)	~20

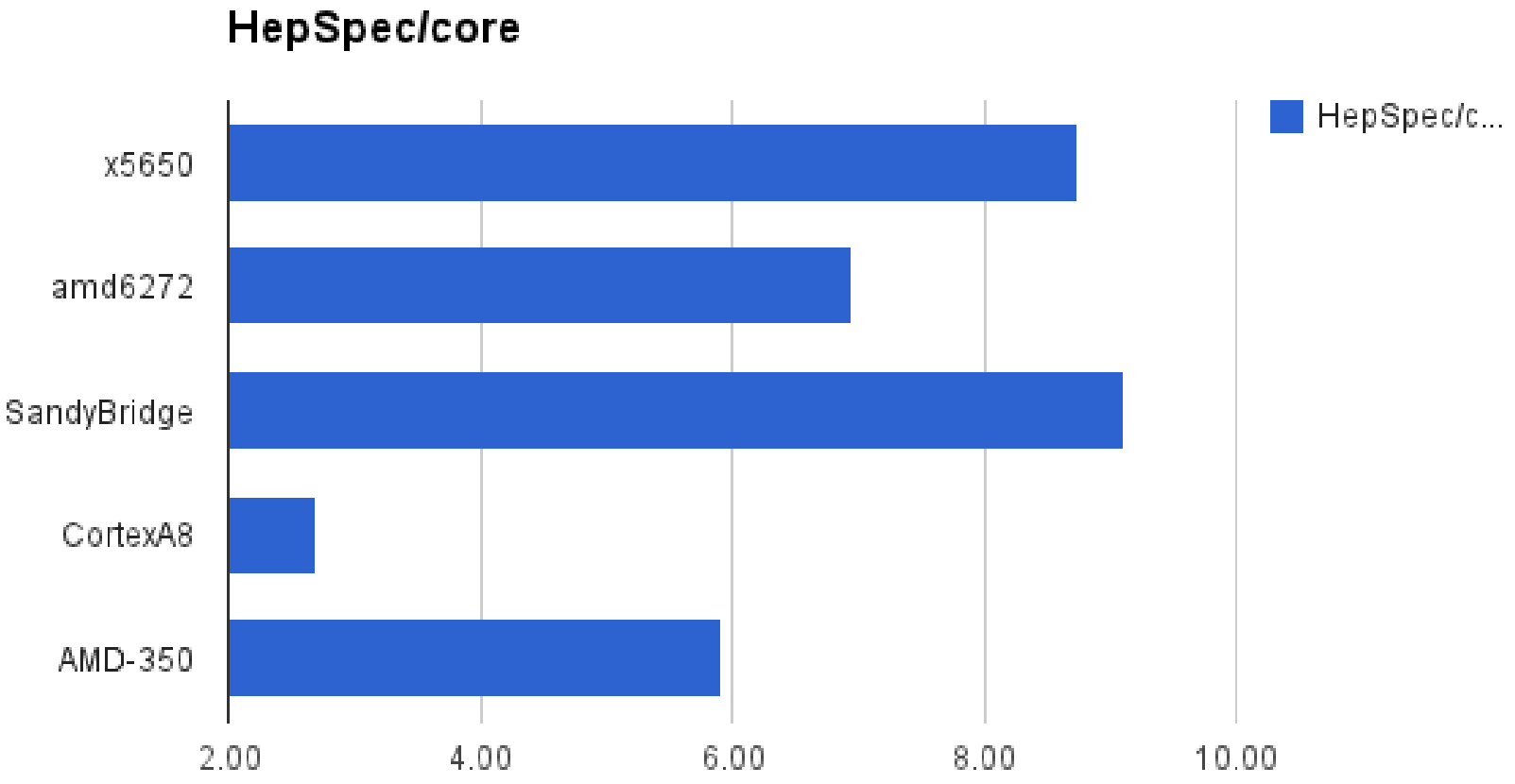
Density | Per-core performance

N.B.: The Intel X5650 is the typical HLT farm node processor

If this ratio continues to hold into 2018 LHCb could do the upgrade with a 600 kW data-centre instead of a new (!) 1 MW one *

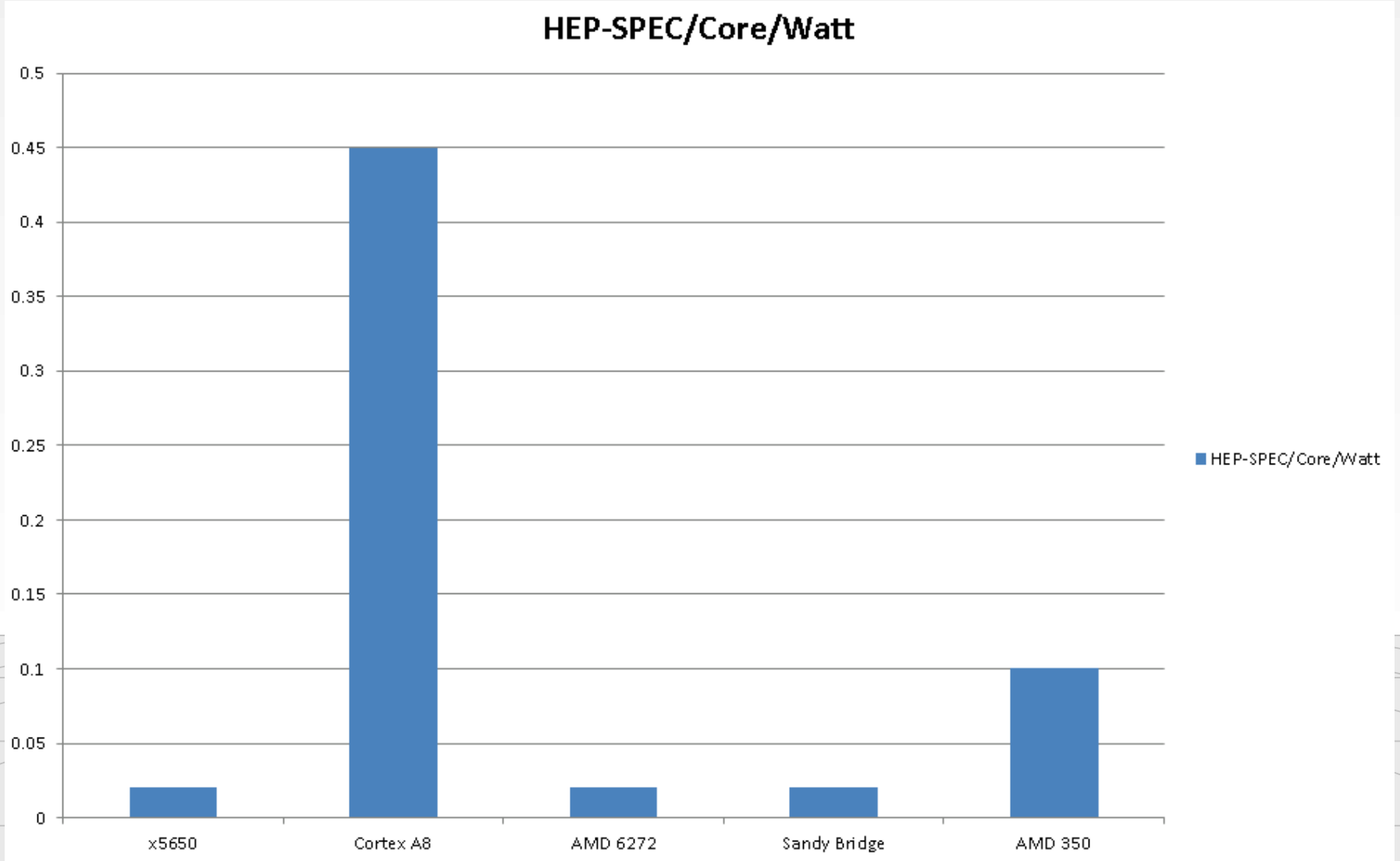
And maybe at some point we need to pay for the power *

How fast is a core?



So we'll need many

How efficient is a core?



Status (April 2013)

Future plans (as of November 2012)

- X-compiler chain completed ✔
- Will now go on to compile stack ✔
- Verification and bench-marking ✔
- Then: full-scale test on fully loaded 192 core system (with a faster ARM – currently use A8 – will have A9 or A15), possibly including real network input (for fun)

In the meantime…

- Increased interest in ARM in and outside HEP: e.g. SC2012, CERN IT will soon provide development machines
- ATLAS & CMS
  https://indico.cern.ch/conferenceDisplay.py?confId=242564
  (Concurrency Forum Meeting – 27/03/2013)
- **LHCb is the first experiment with entire software stack compiled/running on ARM**

Brunel running on ARM

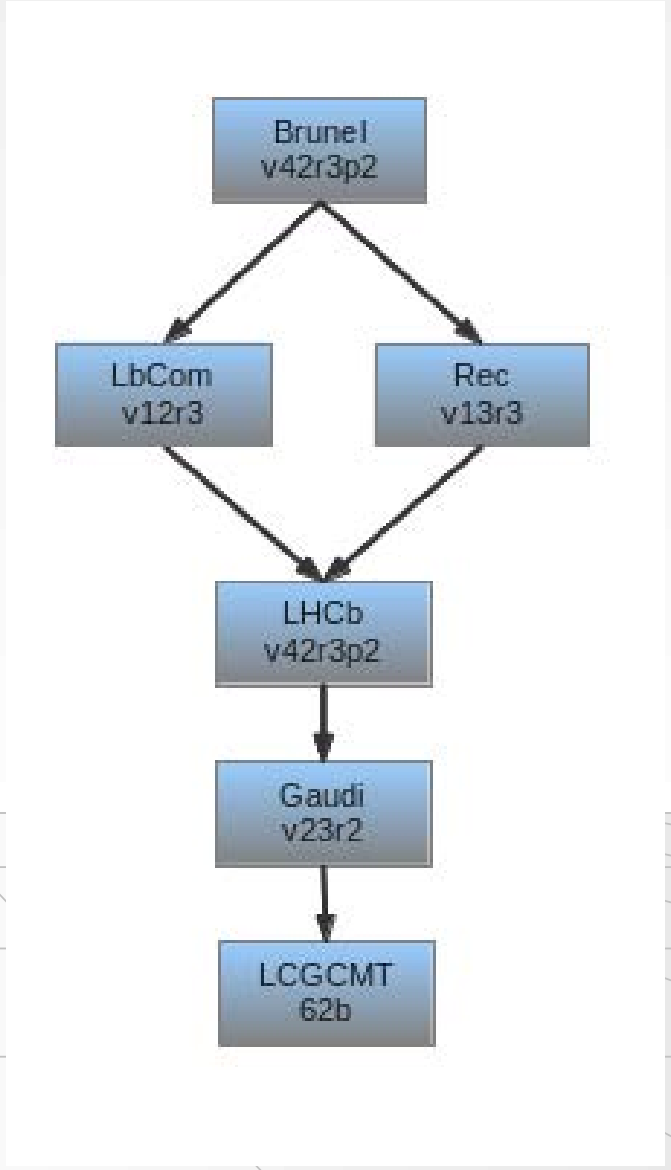
- To speed up development, used CARMA board
  - rather slow compared to Viridis
  - Ubuntu instead of Fedora – slightly different system conf
  - soft-fp (floating point) emulation ☹

- CPU: NVIDIA Tegra 3
  - **Quad-core Cortex-A9 @ 1.3 GHz**
  - **2 GB system RAM**
- GPU: NVIDIA Quadro 100M
  - 96 CUDA cores (not used in this test)
  - 2 GB dedicated memory
- Gigabit Ethernet
- **Ubuntu** 11.04, kernel 3.1.10
  - **No hard-float kernel!**

Brunel running on ARM

- Entire software stack needed by Brunel (except NeuroBayes) has now been compiled on ARM

- Bonus – 'Phys', 'HLT' and 'Moore' Trigger-related projects are also compiled and ready.



Brunel running on ARM

- Multiple Brunel test runs show –
  - almost 100% identical results to x86_64
  - negligible residual differences thought to come from 32-bit architecture (check pending)

"Cluster Match"	sum		mean/eff^*		rms/err^*	
	ARMv7	x86_64	ARMv7	x86_64	ARMv7	x86_64
#calos	81793	81793	98.784	98.784	51.547	51.547
#chi2	1.117573e+09	1.117574e+09	462.97	462.97	292.95	292.95
#links	241390**6**	241390**7**	2915.3	2915.3	3762.5	3762.5
#overflow	897947**2**	897947**1**	10845.	10845.	13205.	13205.
#tracks	84166	84166	101.65	101.65	82.226	82.266

Brunel running on ARM

- Multiple Brunel test runs show –
  - almost 100% identical results to x86_64
  - negligible residual differences thought to come from 32-bit architecture (check pending)

"Brem Match"	sum		mean/eff^*		rms/err^*	
	ARMv7	x86_64	ARMv7	x86_64	ARMv7	x86_64
#calos	50085	50085	60.489	60.489	30.140	30.140
#chi2	2.737109e+09	2.737105e+09	5009.1	5009.1	2866.4	2866.4
#links	546430	546415	659.94	659.92	611.38	611.33
#overflow	403843**4**	403843**0**	4877.**3**	4877.**2**	5074.**2**	5074.**0**
#tracks	586**10**	586**09**	70.78**5**	70.78**4**	48.51**3**	48.51**1**

Brunel running on ARM

- Results of Event reconstruction (1000 events)
  - **Fully within acceptable precision** (as checked against an x86_64 test run)
- Time taken is longer than one would like
  - **~10 hours** (as against **0.5 hours** on a Xeon-server)
- Perspective is important here
  - The **kernel is not built for hard-floats** (neither is the rest of the software) – This is usually a **slowing factor of one order of magnitude** for heavy FP operations
  - The CARMA board is a **dev. kit**, hence slower than the ARM-server that one expects the tests to run on (memory, carrier board etc.)
- Comparable numbers for running time will be with Viridis servers

Immediate future…

- Package the compiled LHCb SW stack for easy deployment for tests
- Revisit cross-compile toolchain
- Get time slice on Viridis servers for real-world tests (and power measurement!)

Side note

ROOT: To patch or not to patch

LHCb patched version

```
000023f8 <ROOT::Cintex::f0a()>:
    23f8:       e92d4800        push    {fp, lr}
    23fc:       e28db004        add     fp, sp, #4
    2400:       e59f3008        ldr     r3, [pc, #8]    ; 2410 <ROOT::Cintex::f0a()+0x18>
    2404:       e59f0008        ldr     r0, [pc, #8]    ; 2414 <ROOT::Cintex::f0a()+0x1c>
    2408:       e12fff33        blx     r3
    240c:       e8bd8800        pop     {fp, pc}
    2410:       fafafafa        .word   0xfafafafa
    2414:       dadadada        .word   0xdadadada
```

CMS patched version

```
00002364 <_ZN4ROOT6CintexL3f0aEv>:
 2364:       e92d4800        push    {fp, lr}
 2368:       e28db004        add     fp, sp, #4
 236c:       e30f3afa        movw    r3, #64250      ; 0xfafa
 2370:       e34f3afa        movt    r3, #64250      ; 0xfafa
 2374:       e30d0ada        movw    r0, #56026      ; 0xdada
 2378:       e34d0ada        movt    r0, #56026      ; 0xdada
 237c:       e12fff33        blx     r3
 2380:       e8bd8800        pop     {fp, pc}
```

ROOT: To patch or not to patch

- **No patch for Cintex**
- Bit-patterns seemed to be loaded much the same way (as without patch)
- "make test-cintex" continues to fail
- Reflex libraries and subsequent "dictionaries" for Gaudi etc. seem to build and run fine (*so far*)

FIN.

BACKUP

Relevance for LHCb

- HEP-SPEC/Coremark as a measure of Online usage is 'synthetic'
- Online workload – VERY specific pattern
  - *n* instances of the same application in parallel, where *n* is the number of cores/hyperthreads
  - No "mixed" work-load – hyperthreading typically adds more in the Online "mono-culture"
- Need to benchmark ARM processors using LHCb code

Status in November 2012

- Cross-compiler toolchain
  - Tested successfully
  - Used successfully to build packages similar to ones needed for the LHCb software stack
- CMT environment, Gaudi + friends
  - Set up using native builds on ARM by Ben and Marco
  - Looks very promising, progressing well
  - Very helpful to have a full fledged Linux system on ARM
- Work in progress
  - The entire benchmarking/testing project is a work in progress
  - The initial figures for processing power look good/comparable in gross to x86-based server solutions
  - Solid comparative numbers for MOORE tasks will give us the full picture
  - More results soon!

Cross-compile Toolchain

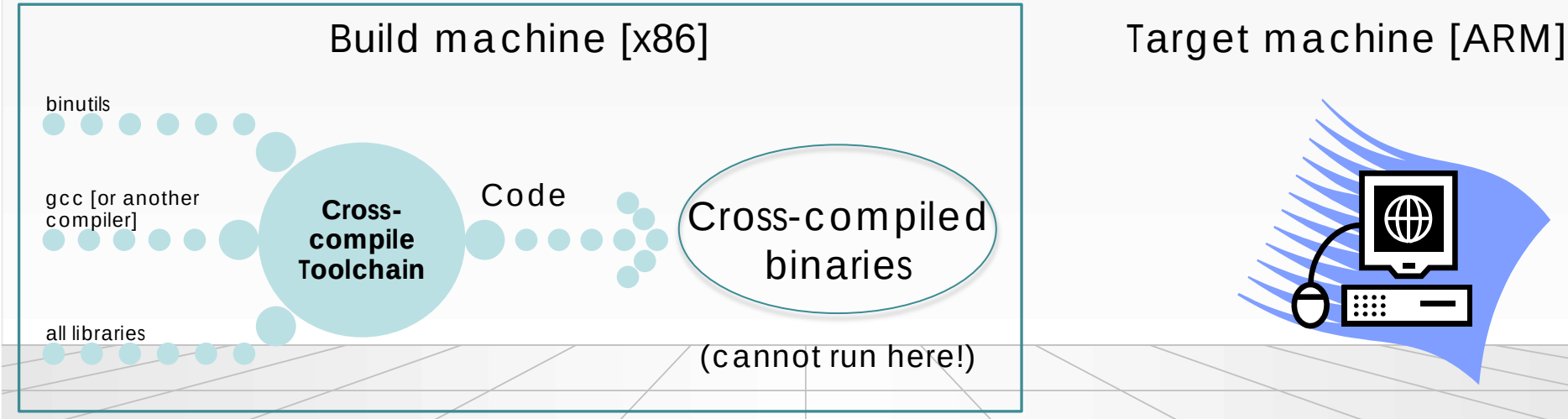
Motivation

- Do not need CMT(and friends) setup on an ARM linux-box
  - Unchartered territory
  - Surprises and roadblocks on the way
  - Ben and Marco working together on this

Cross-compile Toolchain

Motivation

Can use a much faster build machine to compile LHCB software

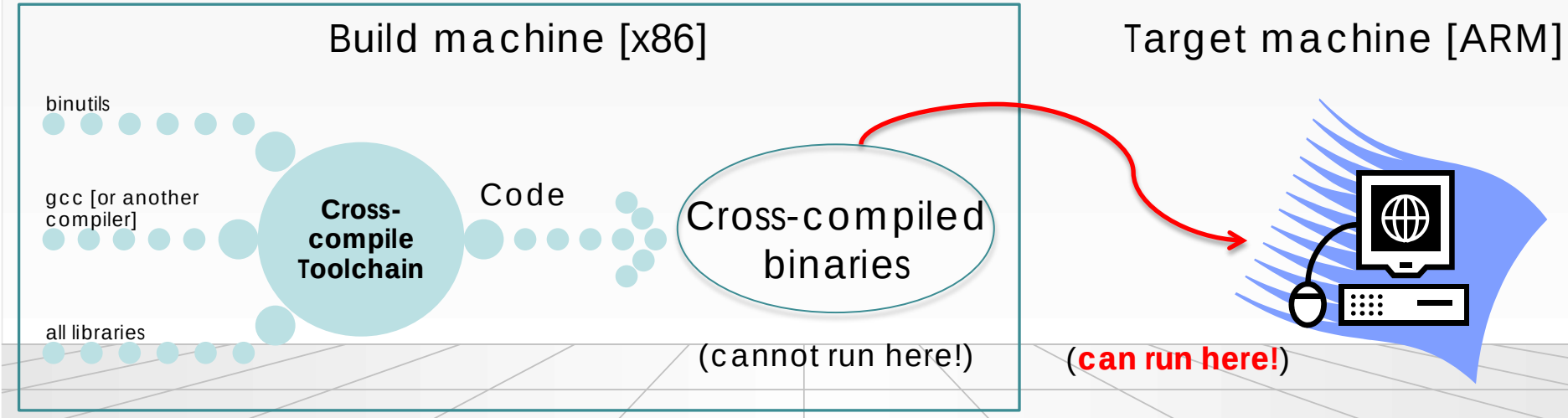


Team (part-time only): Ben Couturier, Marco Clemencic, Niko Neufeld, Vijay Kartik

Cross-compile Toolchain

Motivation

Can use a much faster build machine to compile LHCB software



Team (part-time only): Ben Couturier, Marco Clemencic, Niko Neufeld, Vijay Kartik

Cross-compile Toolchain

- Full cross-compiling toolchain now ready
  - After some sweat and tears to match glibc version with the one used in 'normal' LHCb software builds
  - Current configuration:
    - gcc 4.6.2 cross compiler for ARM-linux with glibc 2.14.1
    - Other relevant libraries (mpfr, mpc, ppl etc.) have also been matched with versions found in /sw/lib/lcg
    - Support kernel versions back to 2.6.25
- Step 1 – simple verification of the toolchain
  - Simple test programs
  - Basic linux tools (tar, bzip etc.)
  - Straightforward cross-compilation
  - Results verified on ARM server (with native tar/bzip etc.)

Cross-compile Toolchain

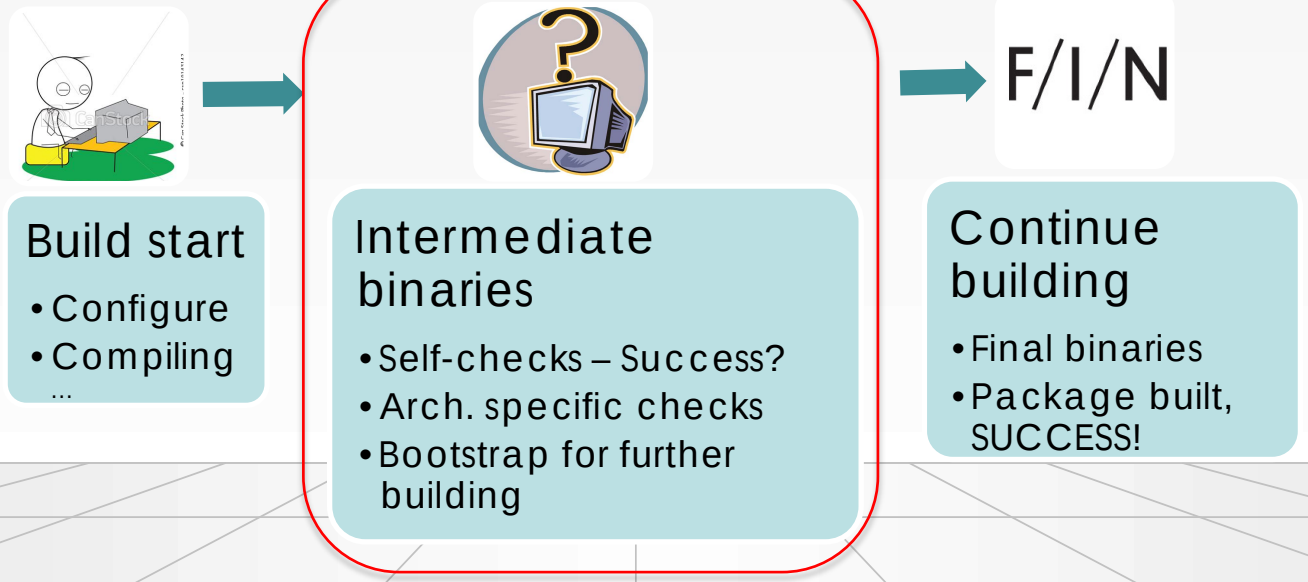
- Step two – more involved cross-compiling

- Try building CMake
- Why?
  - Has generous amounts of C++ code
  - Fairly advanced build process (As does ROOT, by the way)
  - Useful tool if cross-compiled (since Marco is working on using CMake as an alternative to CMT in parallel)
  - Good test of the entire toolchain

- Result?

Fly in the ointment

- Why?
- Almost all non-trivial packages (like CMake) perform "**system introspection**"



- This obviously blows up when a cross-compiler toolchain is used
  - ARM binaries do not run on x86 processors :o

Nifty sidestepping

Goal: Get the intermediate binaries to run

Q. How?

A. *Somehow.*

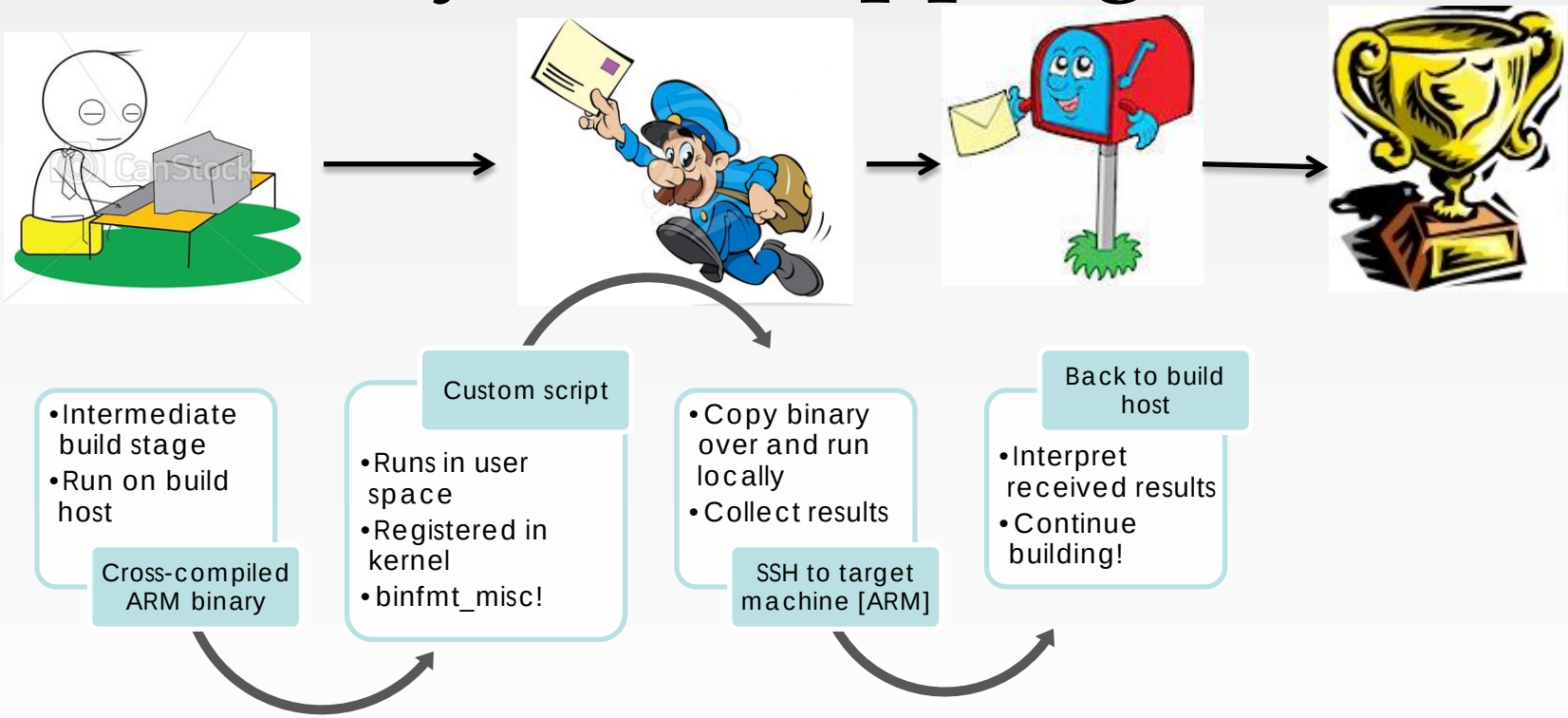
Nifty sidestepping

Goal: Get the intermediate binaries to run

(Thanks Niko!)

BINFMT_MISC

Nifty sidestepping



- From the package build point of view –
  - This is practically transparent
  - Intermediate checks report success
  - Full build goes through

This has now been successfully used to cross-build CMake

Nifty sidestepping

BINFMT_MISC

- Part of the linux kernel
- Can recognize arbitrary executable file formats based on a magic number
- Passes the executable to user space

How?

- Register the executable format (for ARM in our case) in /proc/sys/fs/binfmt_misc/register
- Specify ANY user space application to handle this executable
- Enjoy the goodies