What's New in the LLVM JIT

Lang Hames

October 11th 2023

In November 2021...

In November 2021...

Components:

ORC Core

JITLink

ORC Runtime

In November 2021...

Components:

ORC Core

JITLink

ORC Runtime

Features:

Use regular compilers

Laziness

Concurrency

Out-of-process Execution

Low-level Control

Dynamic loader features

In November 2021...

Components:

ORC Core

JITLink

ORC Runtime

Features:

Use regular compilers

Laziness

Concurrency

Out-of-process Execution

Low-level Control

Dynamic loader features

ORCv2 Deep Dive: https://youtu.be/i-inxFudrgI

Since then...

Since then...

Many new users and contributors — thank you to everyone involved!

Since then…

Many new users and contributors — thank you to everyone involved!

- Platform Coverage

Since then...

Many new users and contributors — thank you to everyone involved!

- Platform Coverage
- Features

Since then...

Many new users and contributors — thank you to everyone involved!

- Platform Coverage
- Features
- Quality

Since then...

Many new users and contributors — thank you to everyone involved!

- Platform Coverage
- Features
- Quality
- Convenience

Since then...

Many new users and contributors — thank you to everyone involved!

- Platform Coverage
- Features
- Quality
- Convenience

Time to talk about MCJIT / RuntimeDyld deprecation

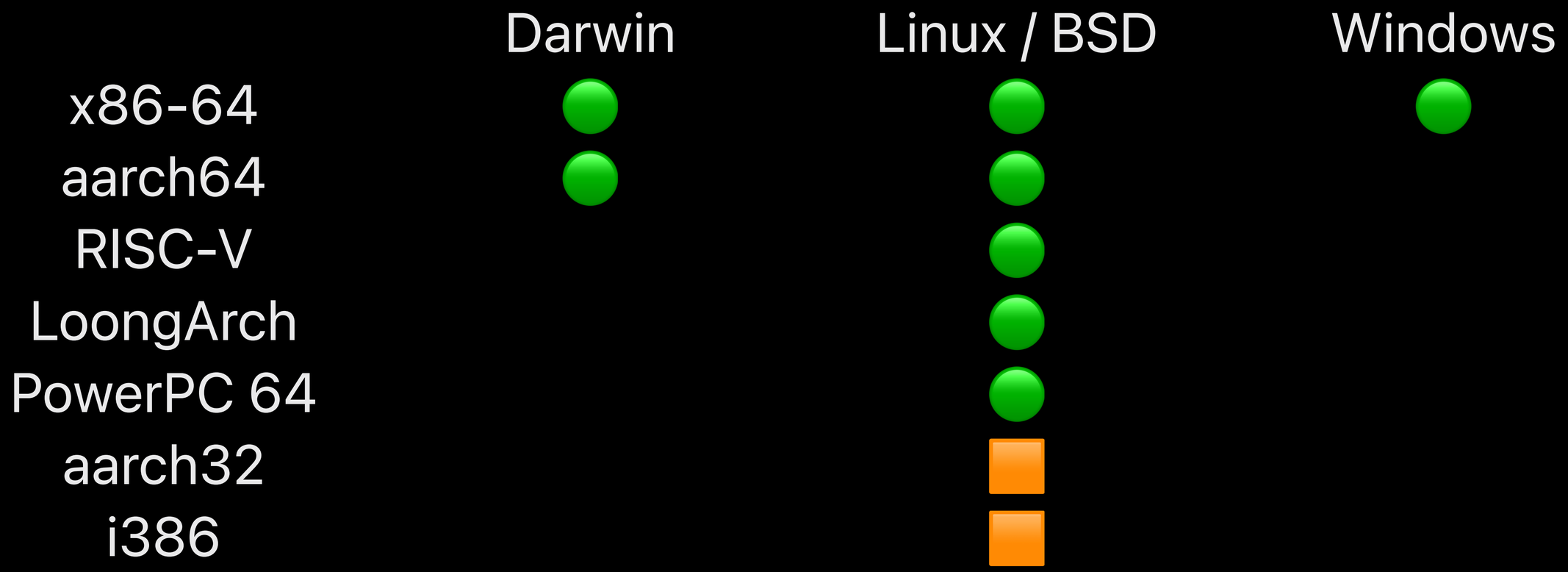
Platform Coverage

In November 2021...

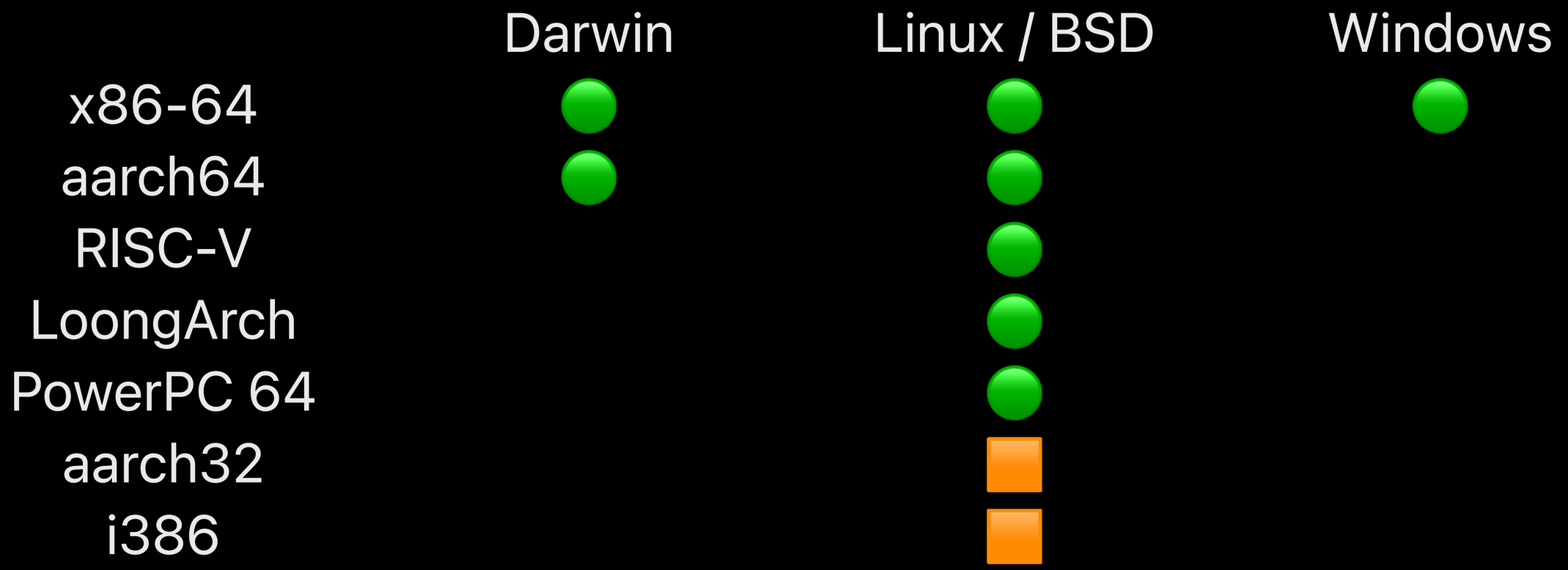
	Darwin	Linux / BSD
x86-64	🟢	🟧
aarch64	🟢	
RISC-V		🟢

🟢 = useable 🟧 = partial support

In October 2023…



In October 2023...



Native Windows JITing in LLVM: https://youtu.be/UwHgCqQ2DDA

In 2024...?

	Darwin	Linux / BSD	Windows
x86-64	🟢	🟢	🟢
aarch64	🟢	🟢	💖
RISC-V		🟢	
LoongArch		🟢	
PowerPC 64		🟢	
aarch32		🟧	
i386		🟧	
BPF		💖	

In 2024...?

	Darwin	Linux / BSD	Windows
x86-64	🟢	🟢	🟢
aarch64	🟢	🟢	💖
RISC-V		🟢	
LoongArch		🟢	
PowerPC 64		🟢	
aarch32		🟧	🤷
i386		🟧	🤷
BPF		💖	
MIPS		🤷	

Features

New Features — Darwin

New Features — Darwin

- Swift and Objective-C support
  - Register language metadata sections using `_objc_map_images` and `_objc_load_image` (same approach as dyld)
  - Swift Extensions and Objective-C Categories now work
  - Swift / Objective-C interoperability is improved

New Features — Darwin

- Swift and Objective-C support
  - Register language metadata sections using `_objc_map_images` and `_objc_load_image` (same approach as dyld)
  - Swift Extensions and Objective-C Categories now work
  - Swift / Objective-C interoperability is improved
- Improved JIT’d code debugging
  - Communicate memory layout changes to debugger via STABS

New Features — Memory Management, Profiling

New Features — Memory Management, Profiling

- `MapperJITLinkMemoryManager` — Anubhab Ghosh
  - Reserve address space up front to avoid out-of-range errors
  - Supports using shared memory for JIT’d code / data
  - See Anubhab’s talk: https://youtu.be/dosXtBAFWiE

New Features — Memory Management, Profiling

- `MapperJITLinkMemoryManager` — Anubhab Ghosh
  - Reserve address space up front to avoid out-of-range errors
  - Supports using shared memory for JIT'd code / data
  - See Anubhab's talk: https://youtu.be/dosXtBAFWiE
- `PerfSupportPlugin` — Prem Chintalapudi
  - Enables Linux Perf profiling for JIT'd code
  - VTune support https://reviews.llvm.org/D146411 (needs new owner)

New Features — Reoptimization

New Features — Reoptimization

- Compile code at a low optimization level

New Features — Reoptimization

- Compile code at a low optimization level
- Re-compile hot code at a higher level and swap in implementation

New Features — Reoptimization

- Compile code at a low optimization level
- Re-compile hot code at a higher level and swap in implementation
- See Sunho Kim’s talk at 4:15pm today!

Quality

Quality

Quality

```
% grep -R report_fatal_error lib/ExecutionEngine | wc -l
      59

% grep -R report_fatal_error lib/ExecutionEngine/{Orc,JITLink} | wc -l
       0
```

Quality

Quality

- Improved error handling
  - Better plumbing, unit and regression tests for error paths

Quality

- Improved error handling
  - Better plumbing, unit and regression tests for error paths
- More async operations, fewer mutexes
  - Definition generator serialization is now via suspension, not mutex

Quality

- Improved error handling
  - Better plumbing, unit and regression tests for error paths
- More async operations, fewer mutexes
  - Definition generator serialization is now via suspension, not mutex
- Reduced library dependencies, code size
  - APIs needing DWARF have been moved to OrcDebugging

Convenience

```
auto J = LLJITBuilder().create();
```

Process Symbols

Before:

```
auto J = ExitOnErr(LLJITBuilder().Create());
J->getMainJITDylib().addGenerator(
  ExitOnErr(DynamicLibrarySearchGenerator::
    GetForCurrentProcess(
      J->getDataLayout().getGlobalPrefix())));
```

Process Symbols

After:

```
auto J = ExitOnErr(LLJITBuilder().create());
```

Process Symbols

Disable:

```
auto J = ExitOnErr(
  LLJITBuilder()
    .setLinkProcessSymbolsByDefault(false)
    .create());
```

Linking Against Precompiled Libraries

```
J.loadDynamicLibrary("libX.so")

J.linkStaticLibraryInto(
    J.getMainJITDylib(), "libX.a")
```

Using the ORC Runtime

- Build compiler-rt:

  Add `-DLLVM_ENABLE_RUNTIMES=compiler-rt` to cmake

```
auto J = LLJITBuilder()
  .setPlatformSetUp(
    ExecutorNativePlatform("liborc_rt.a"))
  .create();
```

Enable debugger support

Link `libLLVMOrcDebugging.a`, call

`enableDebuggerSupport(J)`

Where to next?

Road to MCJIT Deprecation

Road to MCJIT Deprecation

- Platform coverage: Almost there (in many cases better already)

Road to MCJIT Deprecation

- Platform coverage: Almost there (in many cases better already)
- Debugger support: Already better

Road to MCJIT Deprecation

- Platform coverage: Almost there (in many cases better already)
- Debugger support: Already better
- Profiling support: Mostly there (do we need OProfile?)

Road to MCJIT Deprecation

- Platform coverage: Almost there (in many cases better already)
- Debugger support: Already better
- Profiling support: Mostly there (do we need OProfile?)
- Quality: Already higher

Road to MCJIT Deprecation

- Platform coverage: Almost there (in many cases better already)
- Debugger support: Already better
- Profiling support: Mostly there (do we need OProfile?)
- Quality: Already higher
- Stability: Need to be *stable enough*

Road to MCJIT Deprecation

- Platform coverage: Almost there (in many cases better already)
- Debugger support: Already better
- Profiling support: Mostly there (do we need OProfile?)
- Quality: Already higher
- Stability: Need to be *stable enough*
  - Can we define a MCJIT-like subset that we can stabilize?

Get Involved

- JITLink backends — Windows / aarch64, Linux / BPF, others...?
- Profiling support — VTune needs an owner, OProfile still needed
- API design
- Library Layering
- Testing
- Documentation