SIMD Programming in GNU Radio: Maintainable and User-Friendly Algorithm Optimization with VOLK

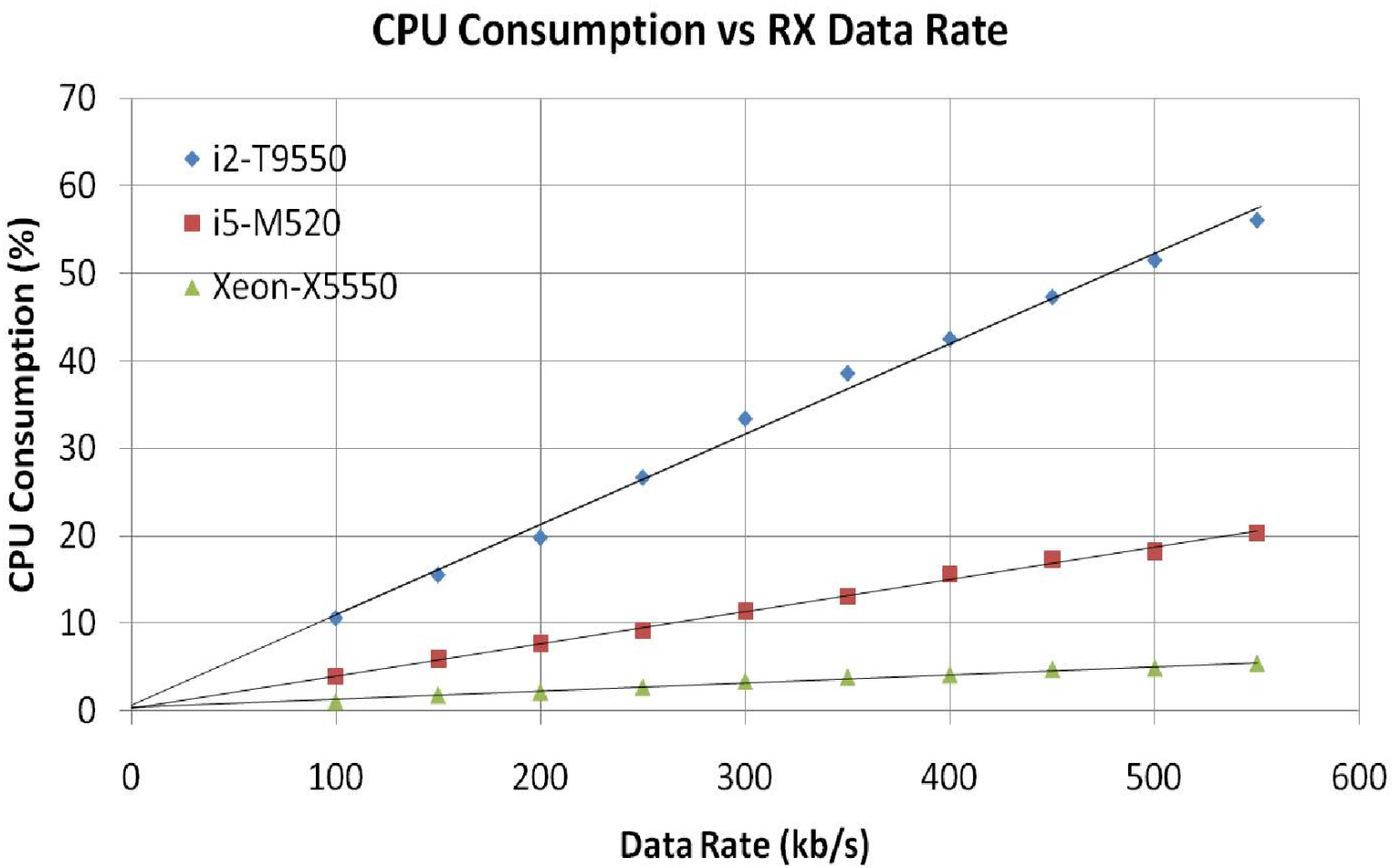
Tom Rondeau (tom@trondeau.com),
Nick McCarthy (namccart@gmail.com),
Tim O'Shea (oshea@umd.edu)

2012-11-30

Outline

I wanna go fast!

What's the problem?



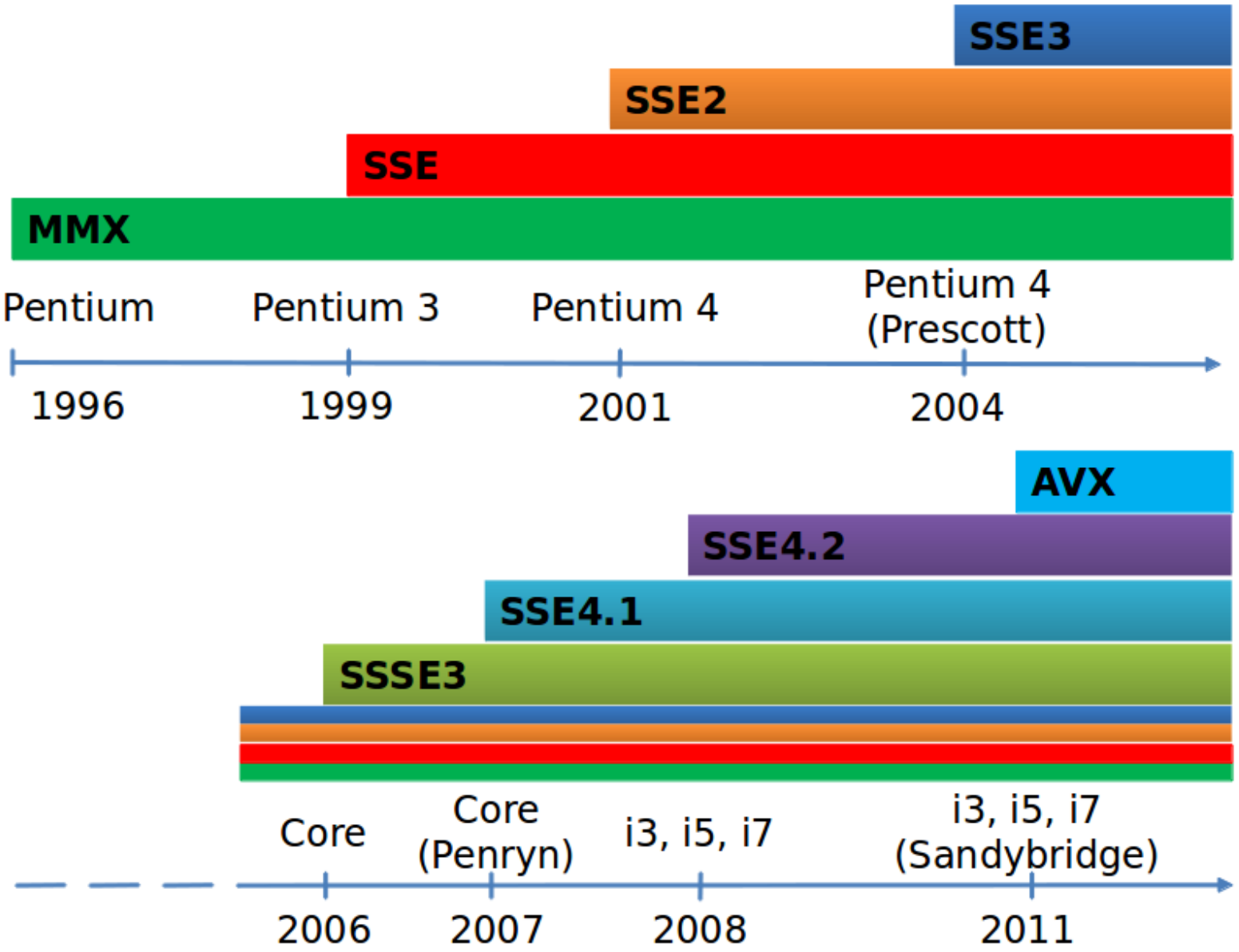
F. Ge, C. J. Chiang, Y. M. Gottlieb, and R. Chadha, "GNU Radio-Based Digital Communications: Computational Analysis of a GMSK Transceiver," IEEE GLOBECOM, 2011.

The SIMD operating space

Implementations of SIMD between processor types and generations.

- Intel: MMX, SSE, AVX
- ARM: NEON
- PowerPC: Altivec
- AMD: 3DNow!
  - Dropped by AMD in 2010; now using Intel's.

SIMD Generations on Intel platforms



Various SIMD extensions make it hard to generalize

- No high-level programming interface that is cross-platform.
  - Intel's intrinsics are a start (between Intel and AMD).
- Still feels like assembly programming.
- Compilers only do so much:
  - GCC handles minor vectorization of C/C++ code.
  - ICC does a better job on Intel platforms.
  - Clang? (sorry, no direct experience, yet).

VOLK Basics

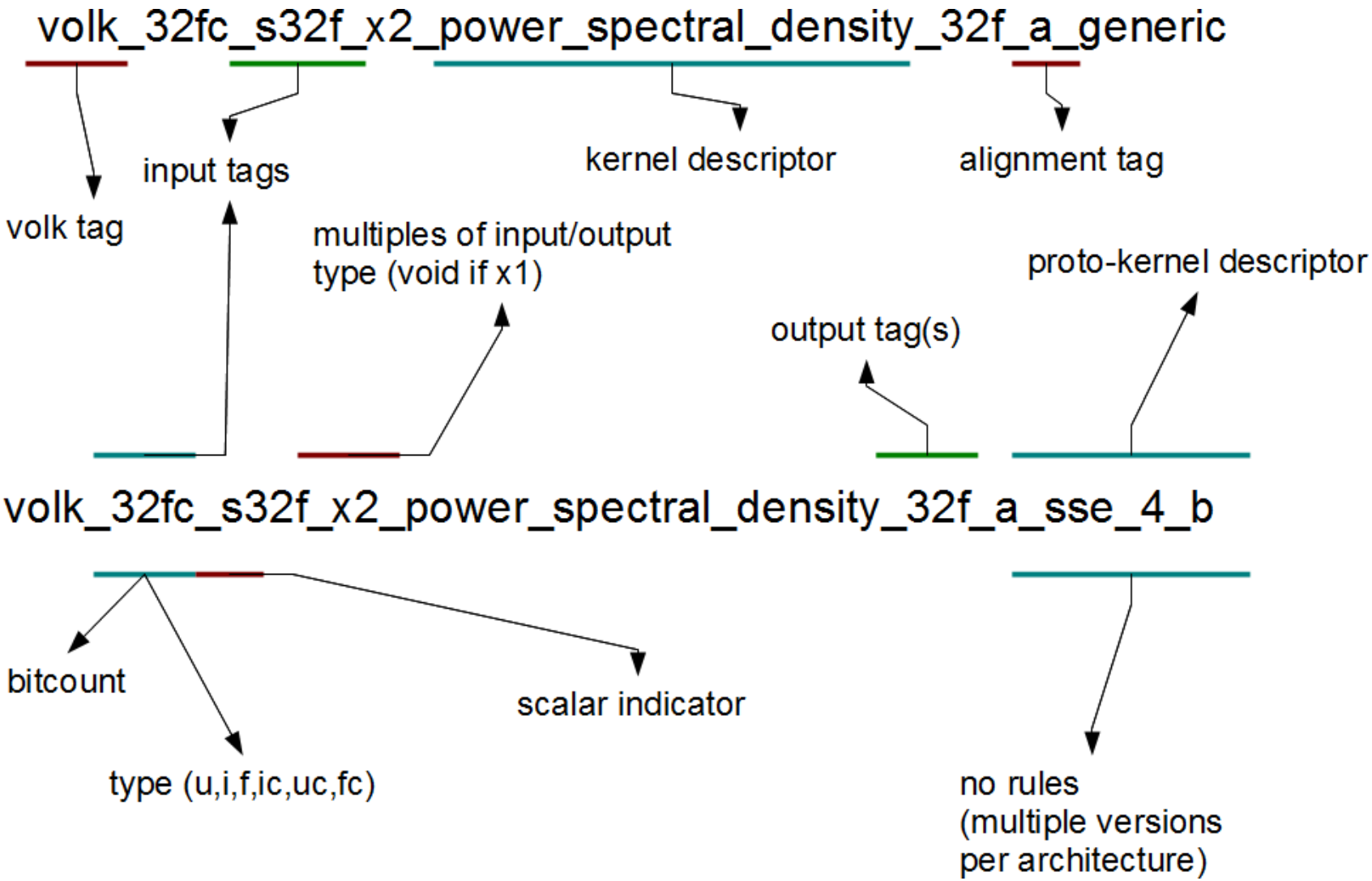
Definitions

- **Kernel**: the abstract unit of VOLK that represents a core mathematical or algorithmic function.
- **Proto-kernel**: platform/architecture/processor specific implementation.
- **Alignment**: required/expected memory alignment for SIMD operation (e.g., 16-bytes, 32-bytes).
- **Dispatcher**: Umbrella of a kernel to properly handle memory alignment issues.

Alignment

- Many SIMD extensions want to operate on memory-aligned vectors:
  - SSE with 128-bit registers expects to operate on 16-byte alignments.
  - AVX with 256-bit registers expects to operate on 32-byte alignments.
- Using unaligned vectors with aligned calls will cause a crash.
- Using unaligned calls tends to incur a performance hit.
- VOLK kernels always have aligned and unaligned versions.
- The dispatcher tests the alignment and calls the right version.

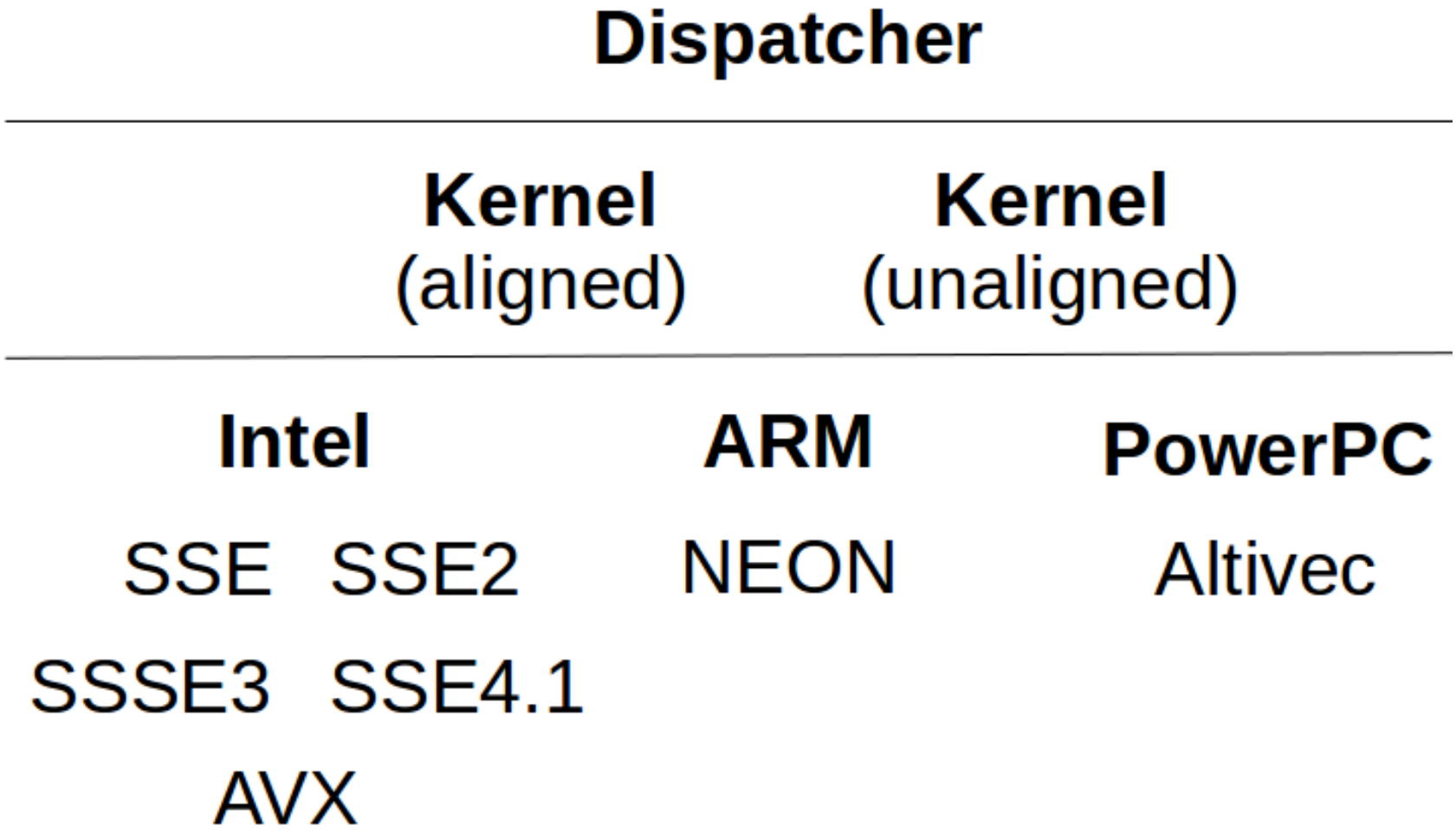
VOLK proto-kernel naming scheme



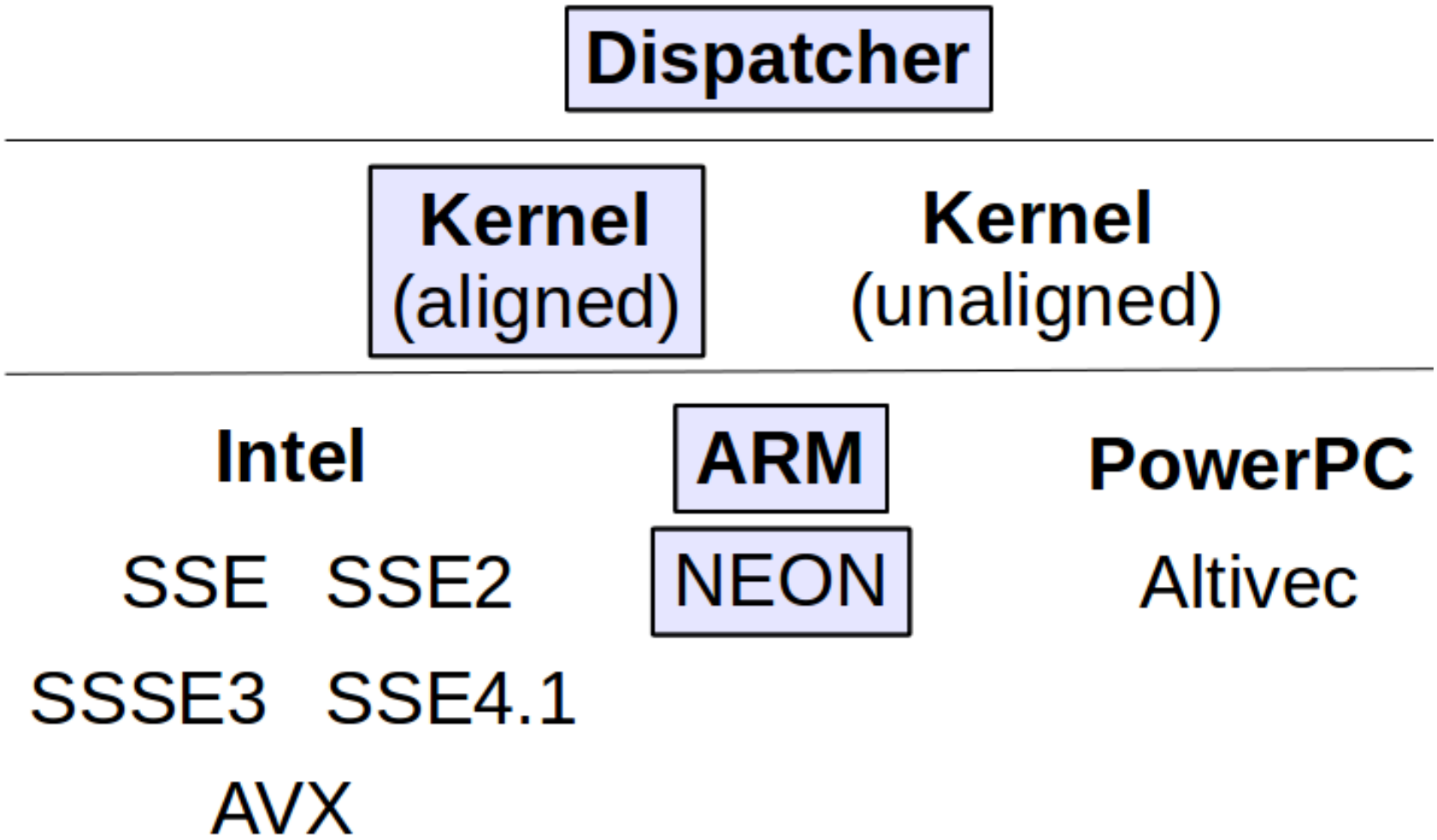
User's Interface to VOLK

- Kernel names are abstracted from proto-kernels and drop final tag.
  - Kernels still have specific alignments.
- Users access via the dispatcher.
- Dispatchers are a further abstraction and do not have alignment.
- Naming example (multiply 2 complex floating point streams):
  - proto-kernel: volk_32fc_x2_multiply_32fc_a_sse
  - kernel: volk_32fc_x2_multiply_32fc_a
  - dispatcher: volk_32fc_x2_multiply_32fc

VOLK Hierarchy



For aligned vectors on an ARM processor, the following path is selected:



Selecting from multiple Proto-kernels

- Some systems have multiple available proto-kernels.
- VOLK must select a proto-kernel for both aligned and unaligned calls.
- Differences in the systems and proto-kernels can make this choice non-obvious.
- By default, VOLK picks the highest version available (AVX > SSE4 > SSE).
- Use `volk_profile` instead.

Using VOLK

An Aside on Memory Alignment

- No real cross-platform support for allocating aligned memory.
- OSes and compilers tend to have different ways of doing this:
  - GCC directive: `__attribute__ (( aligned (x) ))`
  - MSVC directive: `__declspec(align(x))`
    - where *x* must be a constant (#define 16)
  - `posix_memalign(void **ptr, size_t align, size_t s)`
    - obviously, only for POSIX systems.
  - Hand-rolled solution:
    - allocate memory larger than needed; return pointer to first position to meet alignment.
    - must keep track of original pointer to properly free later.
  - Other libraries

Example code

```
#include <fftw3.h>
#include <volk/volk.h>
int main(int agrc, char **argv)
{
  const long int N = 1000;
  const size_t alignment = volk_get_alignment();
  float *a = (float*)fftwf_malloc(N*sizeof(float));
  float *b = (float*)fftwf_malloc(N*sizeof(float));
  float *c = (float*)fftwf_malloc(N*sizeof(float));
  // fill a and b with values to be multiplied

  volk_32f_x2_multiply_32f(c, a, b, N);

  fftwf_free(a); fftwf_free(b); fftwf_free(c);
  return 0;
}
```

Examples: Setup

- CPU: Core i7 870, 2.93 GHz
- Real kernel:
  - `volk_32f_x2_multiply_32f`
  - aligned kernel: `a_sse`
  - unaligned kernel: `u_sse`
  - 400 million samples
- Complex kernel:
  - `volk_32fc_x2_multiply_32fc`
  - aligned kernel: `a_sse3`
  - unaligned kernel: `u_sse3`
  - 100 million samples

Examples: Results (seconds to complete)

Real

	Generic	Unaligned	Aligned
Max	2.422	0.757	0.717
Min	2.399	0.710	0.674
Avg	2.406	0.740	0.701
Var	3.36e-5	1.72e-4	1.48e-4

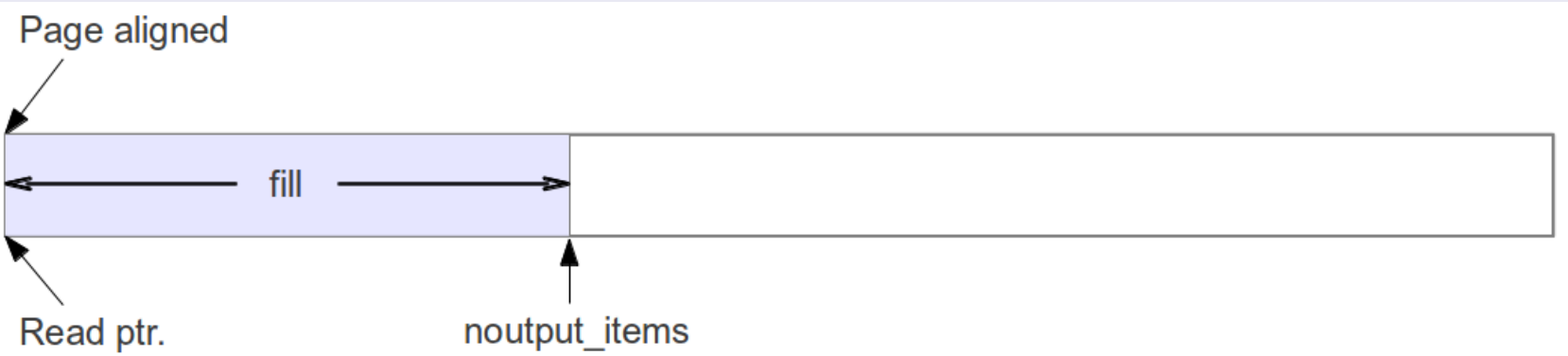
Complex

	Generic	Unaligned	Aligned
Max	2.644	0.409	0.398
Min	2.545	0.368	0.366
Avg	2.568	0.390	0.383
Var	6.33e-4	1.18e-4	7.43e-5

VOLK Issues

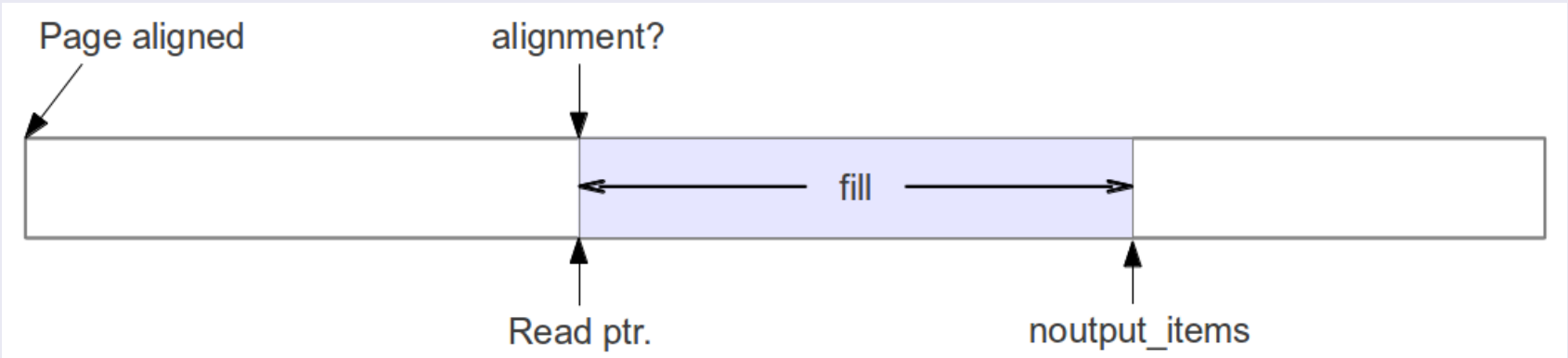
Memory Alignment: SIMD instructions often require some alignment (16 or 32-bytes)

Buffers start page aligned, but the scheduler optimizes for speed and produces noutput_item number of samples to operate upon.



Memory Alignment: SIMD instructions often require some alignment (16 or 32-bytes)

After the first pass, we are not guaranteed a particular alignment.



Memory Alignment: Keeping track of alignment

Scheduler

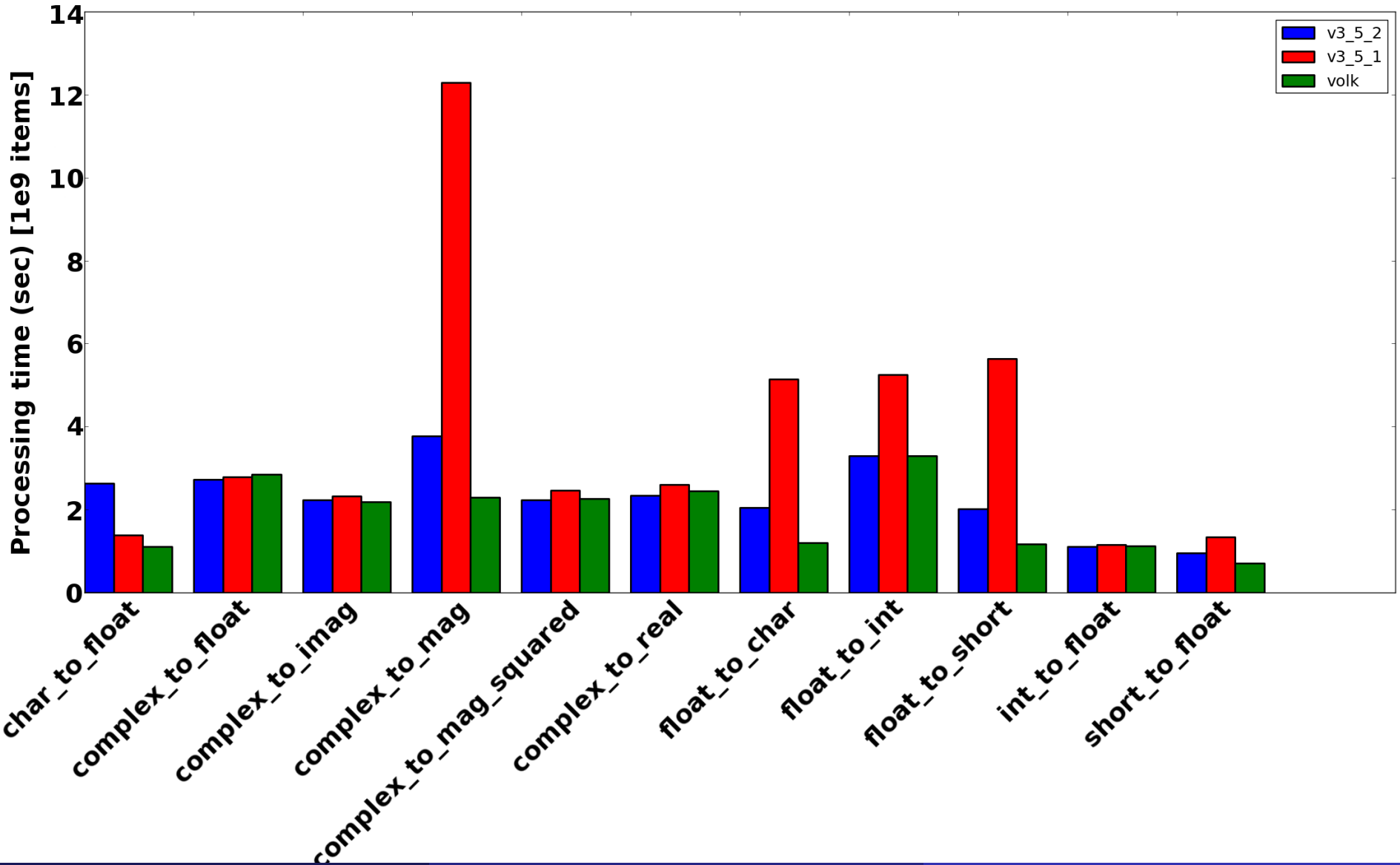
- Scheduler tries to keep alignment but allows unaligned buffers when it cannot do anything else (usually when the number of items is less than the required alignment).
- Works to recovery alignment as soon as possible.
- Passes as large a chunk of memory as it can as quickly as it can – do not reduce throughput or add latency.

Calling the VOLK Kernel

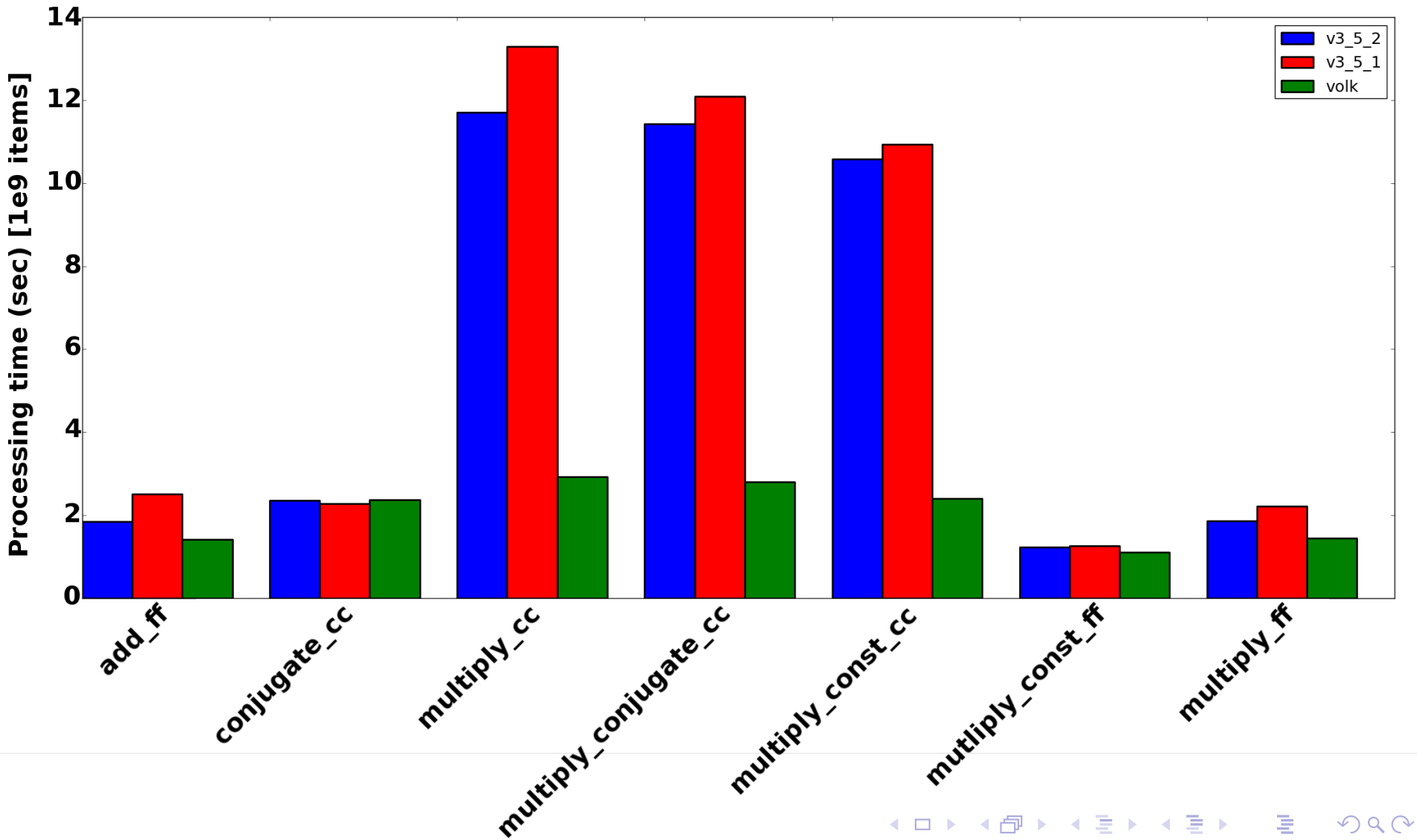
- Generally, just call the dispatcher.
  - This handles the alignment testing and can be optimized.
  - Scheduler still tries to keep buffers aligned.
- Specific calls to test current buffer alignments:
  - `is_unaligned()` returns True if the scheduler passed unaligned pointers.
  - Can still be used in work() function.
  - Generally not necessary now with dispatchers.
- VOLK has `volk_is_aligned()` to test buffer alignment, too.
  - Used by dispatchers.
  - Mostly not needed outside fo VOLK, but it's available.

Results

Current Results: Simple Type Conversions



Current Results: Basic Math



Thank You.

Tom Rondeau (tom@trondeau.com)
Nick McCarthy (namccart@gmail.com)
Tim O'Shea (oshea@umd.edu)