

LIGO-Virgo Collaboration Computing

Peter Couvares

LIGO Laboratory (Caltech)

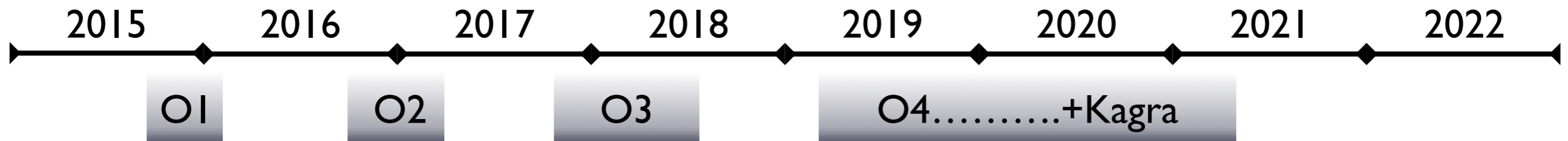
1st ASTERICS-OBELICS Workshop

12-14 December 2016, Rome, Italy.

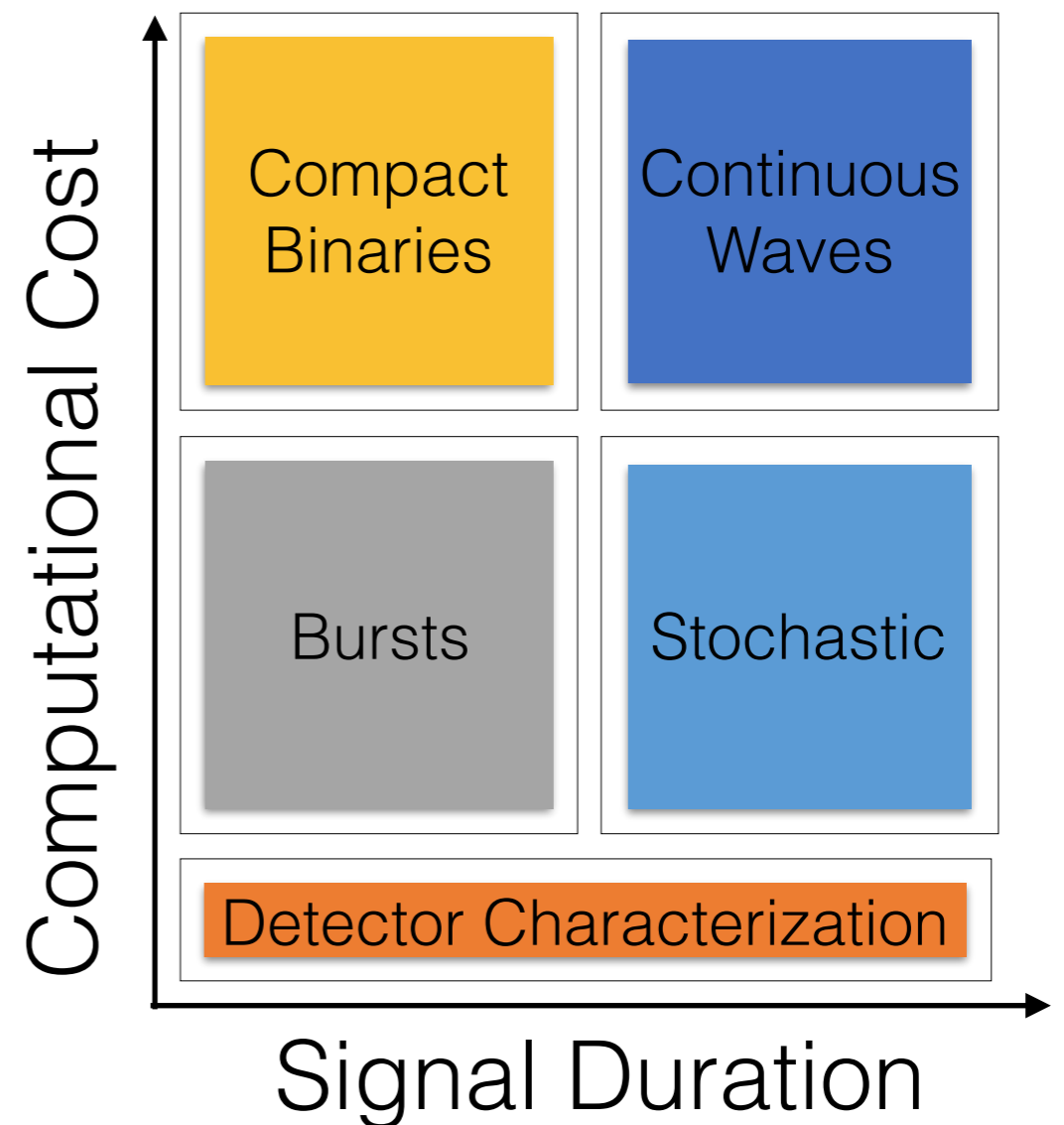


H2020-Astronomy ESFRI and Research Infrastructure Cluster (Grant Agreement number: 653477).

LIGO LIGO/Virgo Data Analysis



- Four astrophysics groups: Bursts, Compact Binaries, Continuous Waves, Stochastic
 - Propose science goals, determine analysis algorithms, write the applications and run them
- Detector characterization group supports the commissioning teams and astrophysics groups
 - Determines analysis algorithms, writes applications, and runs them to identify instrumental artifacts
- Diverse algorithms and methods leading to heterogeneous demands on computing infrastructure



Requirements

- Analyze data in unison from a worldwide network of detectors to reduce background and to enhance scientific output
- Low-latency analysis infrastructure - seconds to minutes
 - Transient alerts and data quality information within seconds of data acquisition
 - Rapid parameter estimation, verification, and follow-up automated (although human vetting is still loop at this time)
- Offline infrastructure - hours to months
 - Detector characterization feedback to commissioning and enhanced data quality generation
 - Deeper and broader searches for transients
 - Searches for continuous and stochastic signals
 - Parameter estimation, model selection and simulations

Executing analyses

- The bulk of our searches are *embarrassingly parallel*.
- All LIGO analyses and computing resources are managed using HTCondor, which schedules work and handles faults to ensure reliable execution of embarrassingly parallel jobs.
 - Broad use of single tools develops a knowledge base in scientific user community
- LIGO and HTCondor team have a very close 15+ year-old collaboration.
 - meetings every 2 weeks between senior staff
 - feature development, big fixes, feedback

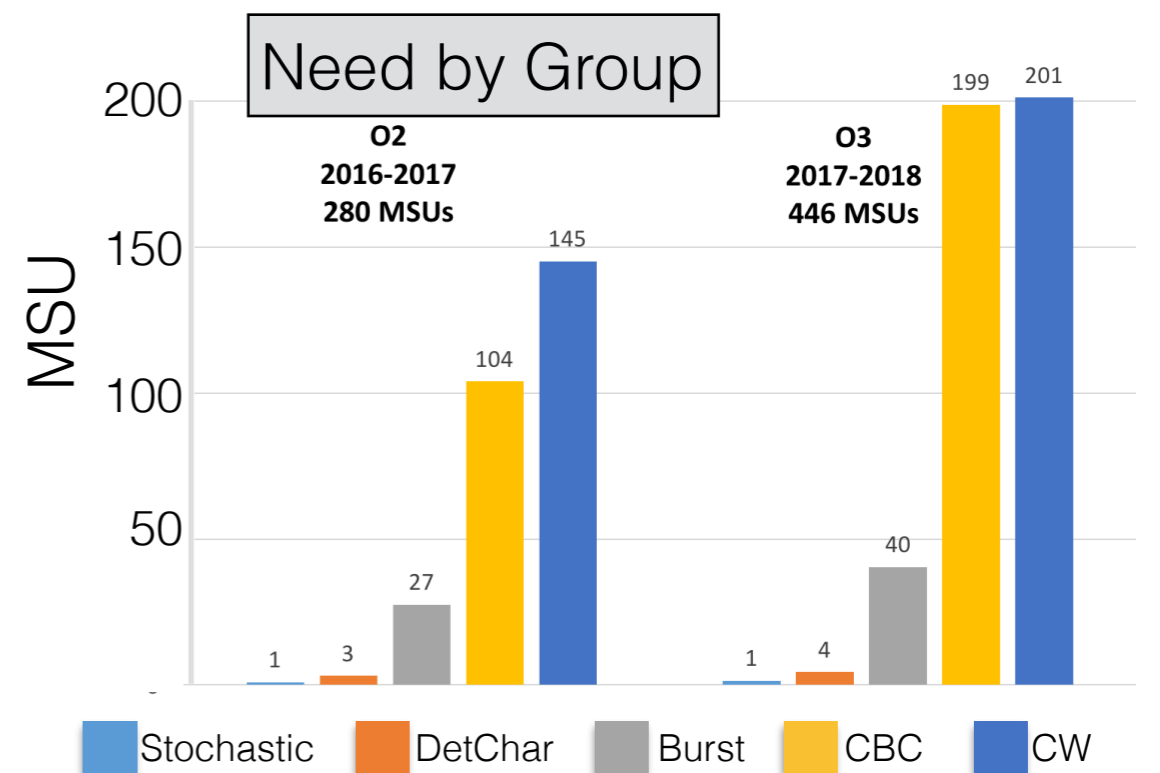
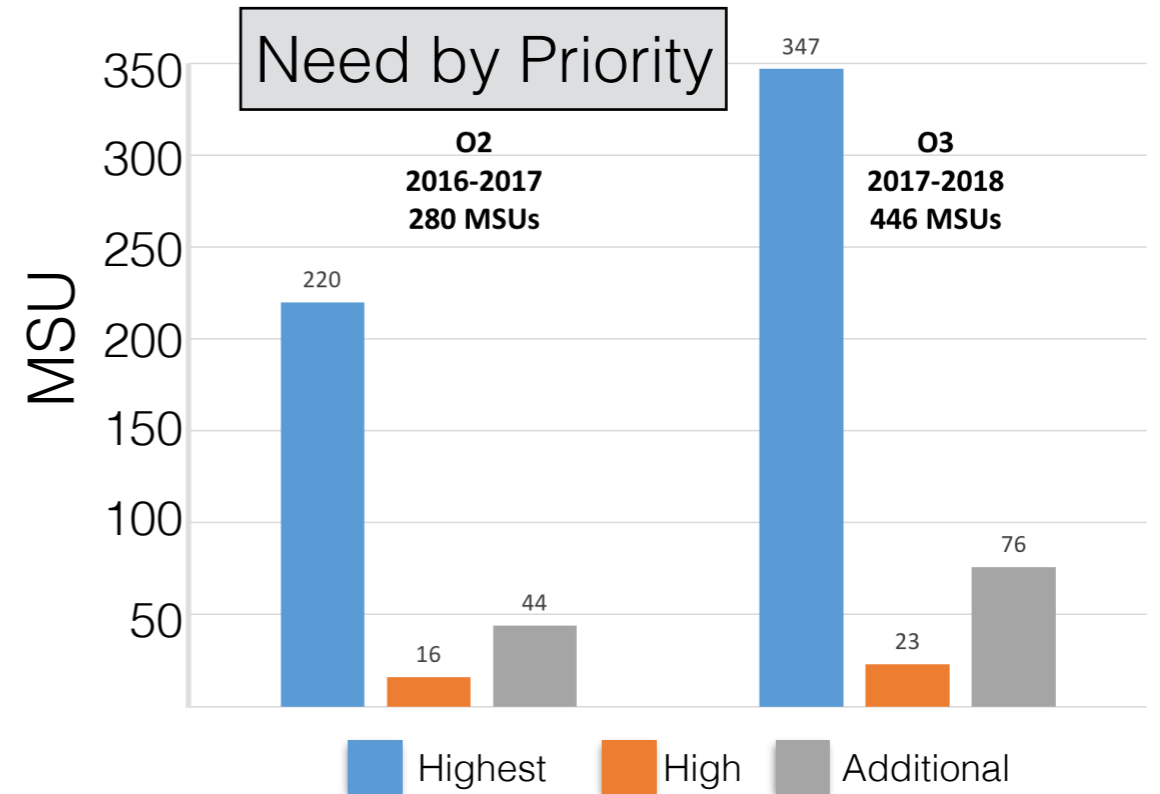


LIGO Implementing analyses

- HTCondor provides the Directed Acyclic Graph Manager (DAGMan) for enforcing dependencies between jobs in a large workflow.
 - Workflows can have $O(10^6)$ or greater individual jobs
 - Extends fault-tolerance from jobs to entire workflows — enables complex workflows to reliably restart from point of failure.
- LIGO continues to use the Pegasus Workflow Management System, developed by the USC-ISI group, that functions as a layer on top of DAGMan for managing data dependencies.
- These tools also facilitate LIGO's use of external resources.

Scale

- Data rates
 - Channels per site: ~200,000
 - Raw and Reduced data: 0.85 PB/yr
 - Strain per IFO: 0.12 MB/s
 - User data: 2.1 PB/yr
- Computing requirements
 - SU=1 core hour on E5-2670
 - O1 actual: 88 MSU (and counting)
 - O3 estimated: ~1/2 billion SU
- Users on LIGO Data Grid
 - ~600 users, ~300 active past year
 - Top 20 users 57 MSU past year

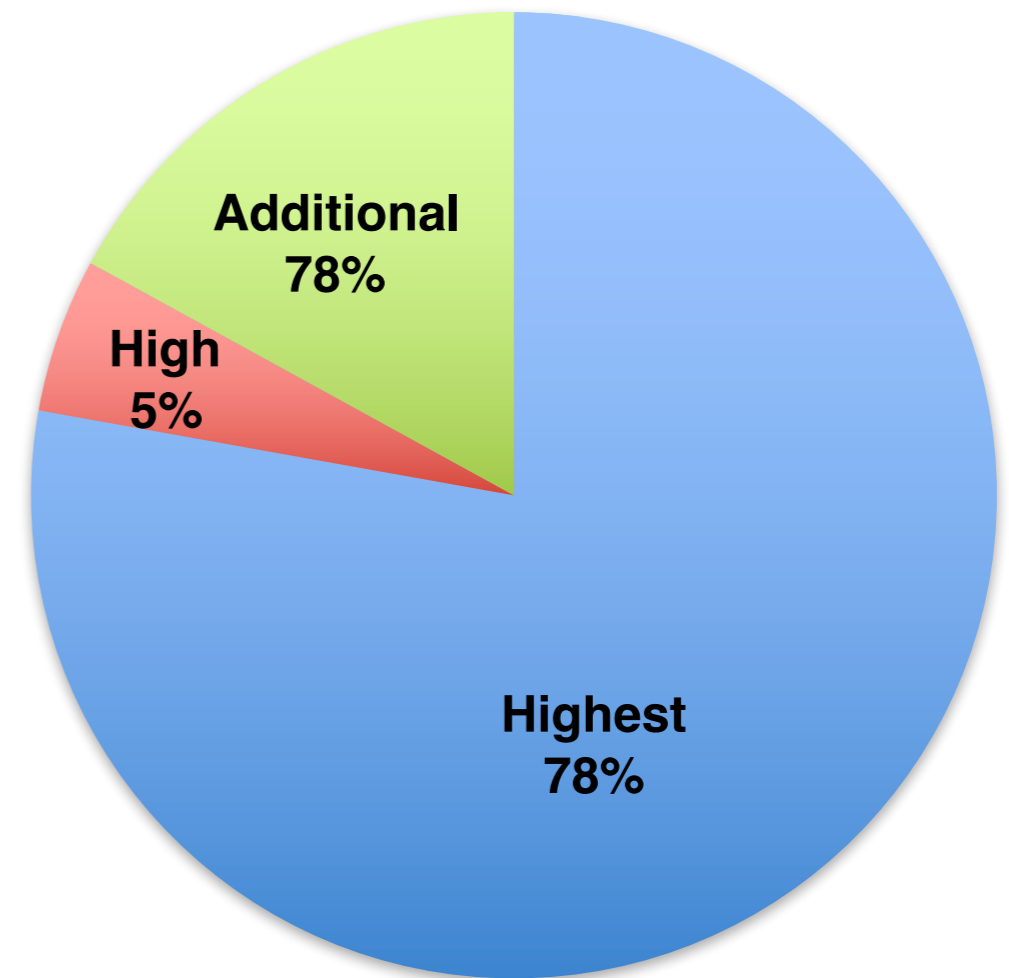


Prioritized Science Goals Define LIGO's Computing Scope

- Three priorities of computing correspond to the priorities of LIGO science goals.
 - *Highest*: critical, core LIGO science. 78% of 2018 (O3) computing.
 - *High*: valuable extensions to astrophysical sources and parameter spaces. 5% of 2018 (O3) computing.
 - *Additional*: higher risk/reward. 17% of 2018 (O3) computing.
- Each planned search is in one of these three categories.

Computing Demand by Priority Category 2018 (O3) Totals

~ 1/2 Billion SUs!



1 SU = 1 aLIGO Service Unit = 1 Intel Xeon E5-2670 2.6Ghz CPU core-hour.



LIGO Hardware Resources: Demand

- LIGO Data Analyses
 - >90 prioritized GW searches and detector characterization analyses.
 - >60 software pipelines implementing them.
- Distribution of computing demand by pipeline: long tail.
 - Top pipeline = 50% of computing demand.
 - Top 10 pipelines = 90% of computing demand.
 - Bottom ~70 pipelines = 10% of computing demand.
- Contrast with distribution of engineering & operations support effort by pipeline: flatter.

LIGO DA Hardware Resources: Supply

- Many types of supply: **dedicated, allocated, opportunistic**. Many providers in the US and abroad:

- Dedicated LIGO Lab clusters (HTC)
- Dedicated LSC clusters (HTC)

LDG

~83%
in O1

- Virgo clusters (mostly allocated on shared resources, HTC)
- PI clusters (shared, HTC and HPC)
- Campus/regional shared clusters (allocated, HTC and HPC) e.g., OrangeGrid, PACE, SciNet
- National shared supercomputers (allocated, HTC and HPC) e.g., XSEDE, Blue Waters
- Opportunistic cycles (campus clusters, DOE labs, HEP clusters, etc.)
- future: commercial cloud (EC2, Azure, Google, Rackspace, etc.)?

OSG

~17%
in O1

- Two runtime software environments: LIGO Data Grid, Open Science Grid

Analysis Metrics

- LIGO worked with Condor team to extend HTCondor to enforce requirement that all jobs be “tagged” with strings that identify their scientific goal.
- Each night, usage statistics are collected from LDG clusters and partners in VIRGO
- LVC accounting system’s web interface allows all LVC members to understand historical usage by cluster, search, and user, and time period.
- Allows us to measure usage by each search and compare to pre-run estimates
- All LIGO jobs in O1 were tracked! Accurate statistics by science goal, search pipeline, activity (development vs. testing vs. production), and observing run.

Computing Optimization

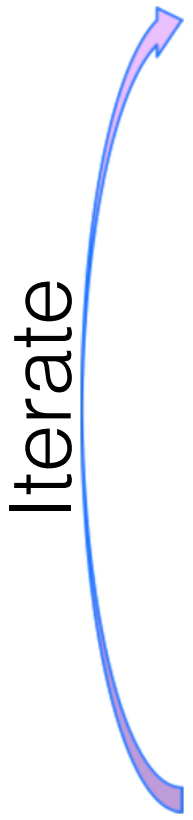
- Early LSC estimates for Advanced LIGO data analysis computing needs were very high (billions of CPU core-hours), exceeding projected computing resources.
- Funding agency (NSF) alarmed, and charged LIGO with two tasks: prioritize LVC science (implicitly, in case something needed to be cut), and demonstrate that our codes were extraordinarily efficient.
- It turns out they were efficient, but not extraordinarily so given their scale.
- LIGO Laboratory organized a multi-year effort led by a new Data Analysis Computing Manager position at Caltech, dedicated optimization staff, and the regular attention of LVC management, working group chairs, and search leads.

The LIGO logo is located in the top-left corner of the slide. It consists of the word "LIGO" in a bold, black, sans-serif font, followed by a series of concentric, light gray circles that resemble a ripple effect or a stylized gravitational wave pattern.

LIGO Computing Optimization

- In lead-up to aLIGO, NSF insisted LIGO place a renewed emphasis on efficiency and optimization of computationally expensive data analysis pipelines.
- The LIGO Optimization Team is a joint effort between LIGO Lab and LSC of between 2 to 5 FTEs over time. One FTE at Caltech currently open!
- pyCBC and GstLAL CBC search pipelines were the two largest consumers of computing resources in aLIGO.

Optimization Approach: “The Whole Patient”

- 
- Scientific Prioritization and Scoping
 - Estimation and Benchmarking of Computational Costs
 - Optimization of Data Analysis Methods and Algorithms
 - Optimization of Code Implementation and Libraries
 - Compiler Optimizations
 - Workflow Management Optimizations
 - Development, Testing, and Simulation Process Optimizations
 - LIGO-Virgo Computing Network Scheduling Optimizations
 - Resource Supply Optimizations (make more cycles available)
 - Workflow Portability Optimizations (expand usable resources)
 - Hardware Procurement
 - Pipeline Reviews including Computational Efficiency
 - Documentation, Training, Collaboration and External Engagement

Neglect nothing, but focus on “bang for the buck” and where optimization effort can be most effective. Avoid burdening scientists when the payback is small.

Results

- Dramatic gains in efficiency and estimated computational cost between May 2014 and May 2015:

Observing Run	CBC High Latency Deep Detection Search			CBC Low Latency Detection Search		
	2014	→	2015	2014	→	2015
Estimated in:	2014	→	2015	2014	→	2015
O1 (2015–16)	115	→	5	5	→	0.5
O2 (2016–17)	369	→	39	27	→	4
O3 (2017–18)	896	→	133	104	→	17
Total	1380	→	177	136	→	22

Values are in Millions of aLIGO Service Units (MSU), where 1 SU = 1 Intel Xeon E5-2670 2.6GHz CPU core-hour.

- Factor of ~ 8 reduction in estimated computational demand of high-latency CBC search.
- Factor of ~ 6 reduction in estimated computational demand of low-latency CBC search.
- Possible additional factors of 2-4 reduction under development for offline CPU code, order of magnitude for offline GPU code.

Resource Optimization (Supply-Side)

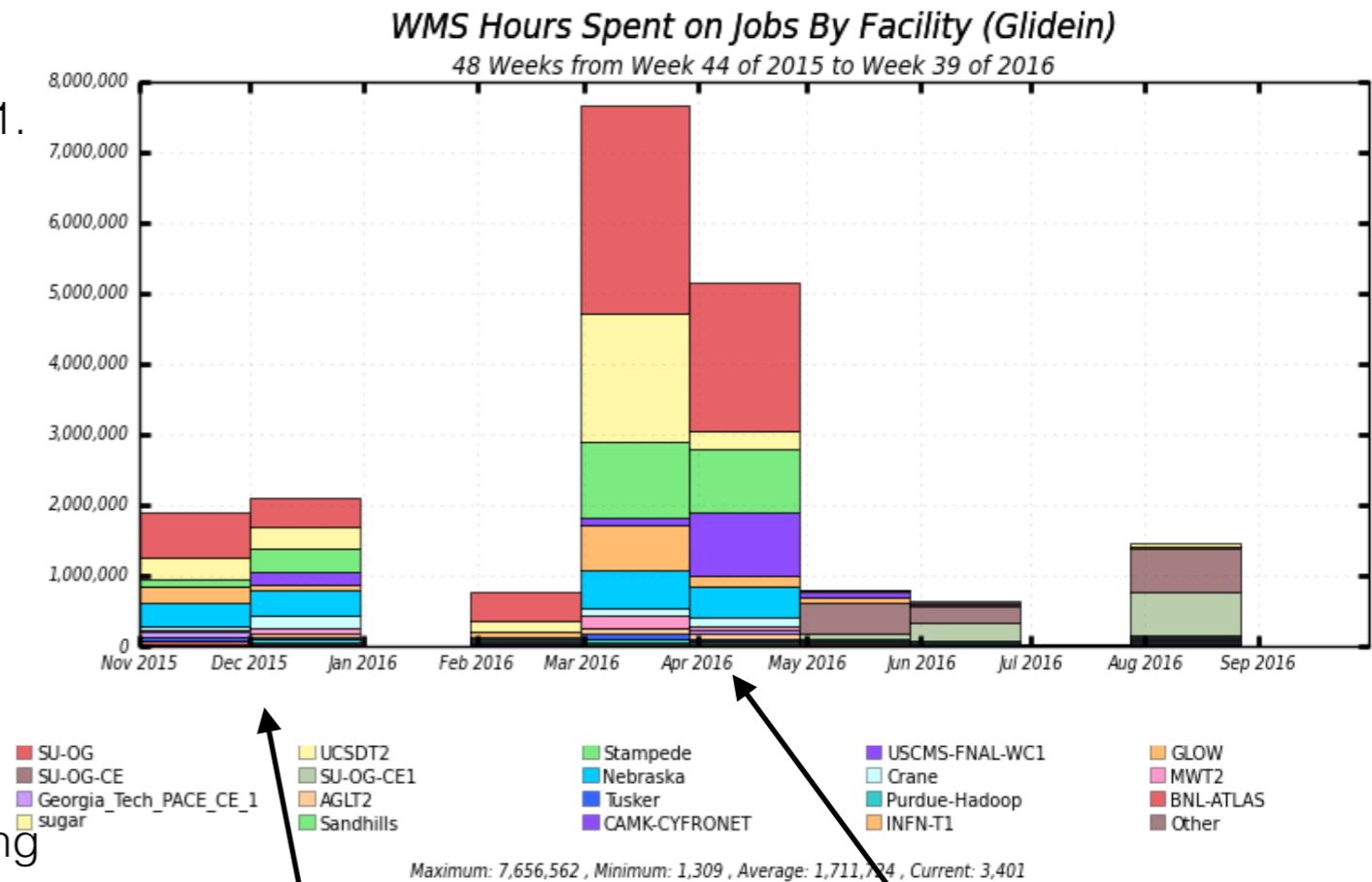
- **Open Science Grid – outstanding initial results.**
- OSG's primary value to LIGO is in providing cyber-infrastructure (“plumbing”) and NSF-funded staff to help enable LIGO to harness available non-LDG resources (campus and regional clusters, PI clusters, HPC centers, XSEDE resources, Virgo clusters, and even future cloud resources) that we've been unable to effectively utilize in the past.
- There are also a non-trivial amount of idle CPUs to be harnessed opportunistically at non-LIGO OSG institutions — that's not our main target, but it's a nice side benefit.
- OSG is not a replacement for LDG clusters providing baseline supply and low-latency computing — rather it's a “universal adapter” to external resources, and **a means to provide elasticity to our computing resources, to meet peak (or unexpected) demand.**
- In O1, LIGO harnessed >14 MSUs from external resources via OSG, freeing LDG clusters for other analyses. In total, 17% of O1 computing was delivered by OSG, which is more than any individual LVC provider except AEI-Hannover.

Shared Resources

- **Given demand estimates, LIGO/Virgo are unlikely (but not certain) to need substantial shared computing resources through 2017-18 (O3) to meet our science goals.**
- However, modest shared resources will be of benefit for short-term demand spikes, new searches, and hardware trade studies.
- We have engaged with XSEDE (US Supercomputer/HPC Network):
 - So we're prepared to scale quickly if we need to.
 - So we can continue to leverage XSEDE HPC expertise (ECSS).
 - So we're ready for (and can help define) evolving HTC/HPC computing models for future LIGO data analysis, post-O3.
- We have engaged with the Open Science Grid (OSG):
 - To bridge the LIGO-Virgo Computing Network (LVCN) to shared computing resources available to LIGO (including but not limited to XSEDE).
 - To enable sharing of short-term LVCN surpluses.

LIGO Use of OSG

- Production offline CBC analysis utilized OSG in O1.
- The LIGO analyses running across >20 different OSG resources.
- >20 million OSG CPU-hours for O1.
- About 1/3 of these OSG cycles were provided by LIGO-connected partners (e.g., the Syracuse campus grid), about 1/3 by XSEDE, and the remainder were opportunistic cycles scavenged from LIGO-unrelated clusters around the US.
- ~5TB of input data stored at the Holland Computing Center (HCC) at the University of Nebraska-Lincoln.
- The total data volume distributed to jobs from Nebraska >1PB.
- Data rates from Nebraska storage to worker nodes ~10Gbps sustained. (Recently demonstrated >30Gbps!)



GW150914 CBC
offline running

GW151226 CBC
offline running

PyCBC GPU Search

- Offline search for binary black holes, binary neutron stars, and neutron star—black hole binaries
- Searches across a pre-constructed bank of signal parameters
- Compute coincident detection-statistic for candidate events
- Measure noise background to determine candidate event significance

PyCBC Search

- Pipeline cost dominated by filtering stage:
 - Correlate and matched filter
 - Chi-Squared signal-based veto
 - Template Generation and Event Finding
- Pipeline described in S. A. Usman et al. arXiv: 1508.02357 (Classical and Quantum Gravity)

Search Algorithm

```
for binary parameters in pre-generated template bank:
```

```
    generate waveform template(f;binary parameters)
```

```
for d in data:
```

```
    correlate(template,data),  
    inverse FFT to get signal-to-noise ratio,  
    peak find and cluster
```

```
if (peak above threshold):  
    signal-based veto
```

Graphics Processing Units

- The FindChirp algorithm is can be efficiently implemented on GPUs, if the the following practices are followed:
 - Move data over PCIe bus to card once at start of computation
 - Minimize coalesced reads across the GPU's memory bus
 - Take advantage of “black box” CUDA FFT library
 - For LIGO data analysis, we can use single precision for fast computation on cheap consumer cards

PyCBC on GPUs

- Use the same search engine replacing CPU code with CUDA kernels
- Easy of maintenance and allows mixing of CPU and GPU jobs in a workflow

for binary parameters in pre-generated template bank:

```
generate waveform template(f;binary  
parameters)
```

```
for d in data:
```

```
    correlate(t,d), cuFFT, peak find,  
    cluster
```

```
    if (peak above threshold):
```

```
        signal-based veto
```

2015 GPU Trade Study

- Determine throughput based on different data types (clean, glitchy) and compare to CPU throughput
- Focused on Maxwell architecture using consumer-grade cards

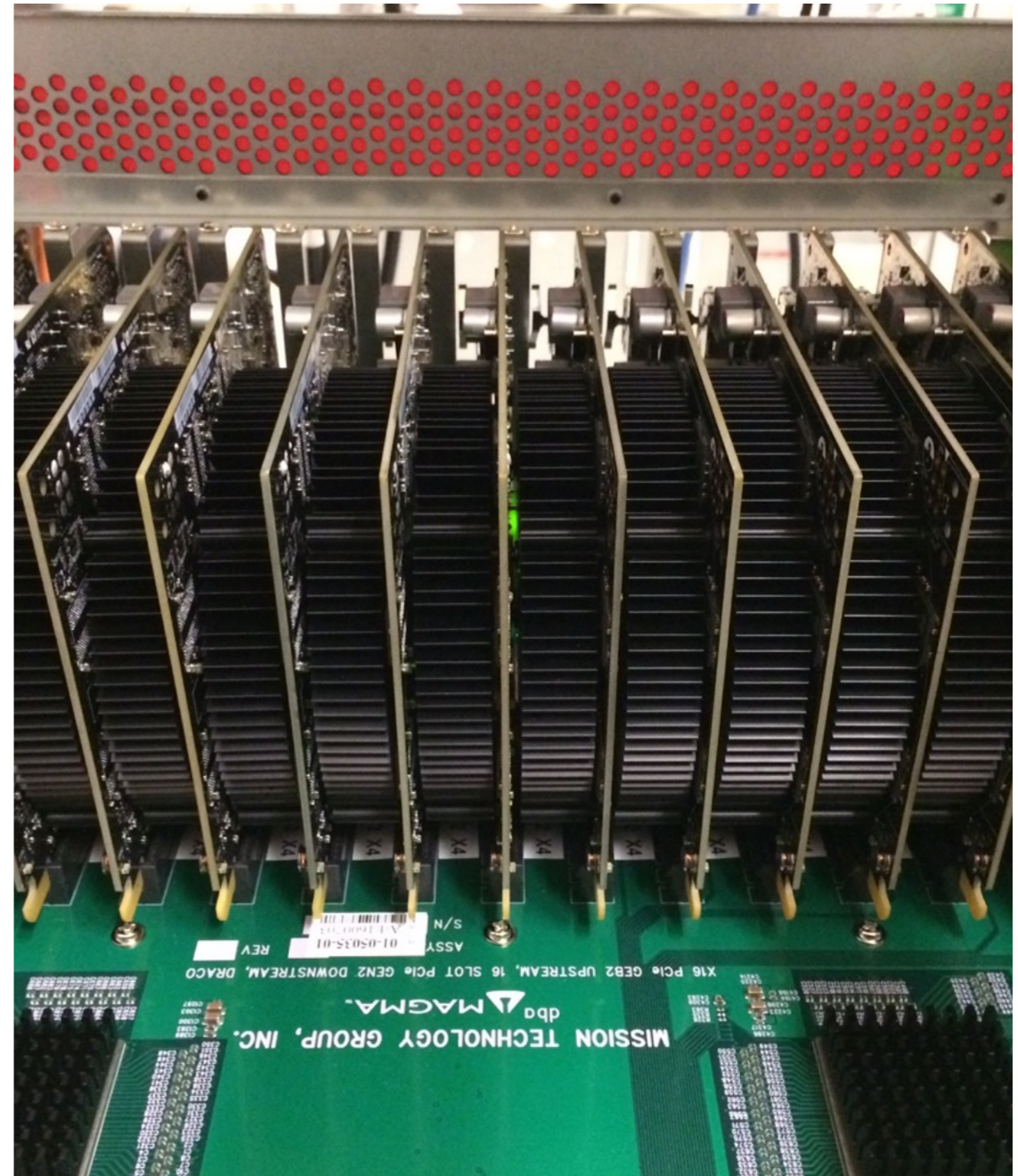
Hardware	Best Data	Worst Data	Cost	Ave templates/\$
E5-1660 v3	76,800	59,200	\$1100	65
E3-1220 v3	29,600	25,900	\$205	138
GTX 980	221,000	213,800	\$550	360
GTX 750 Ti	120,700	116,600	\$140	780

Current Template Throughput

- Test machine with 8 x GTX750 Ti Cards
- Using CUDA 7.5 and improvements to data handling in PyCBC we can presently get 160,000 templates per GTX 750 Ti
- Compare to 2015 average of 110,000 templates per GTX 750 Ti
- The GTX750 Ti are very promising as a co-processing technology for adding additional compute power to racks, or for stand-alone GPU systems (Maxwell, no additional power, cost \$120 each)

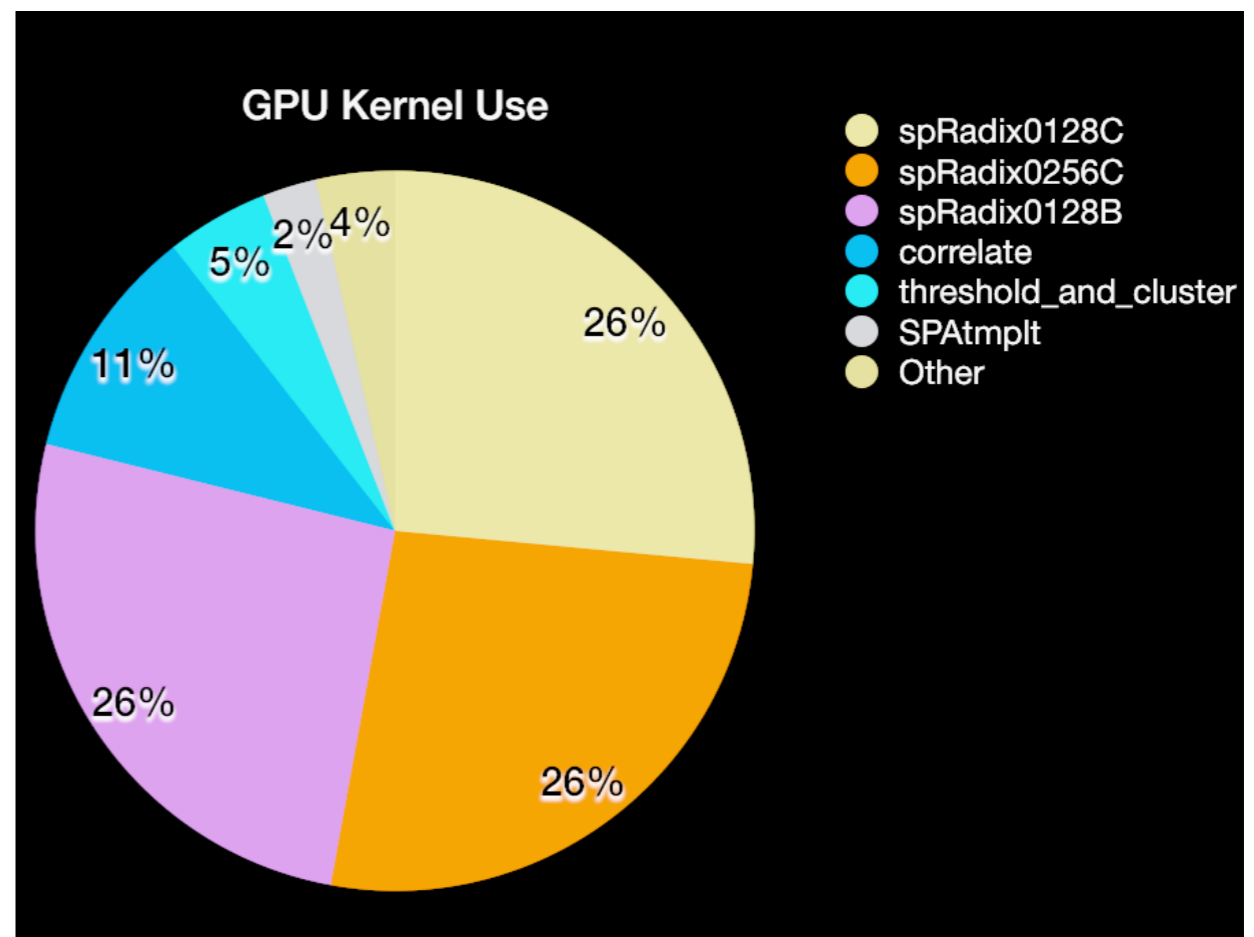
LIGO Syracuse GPU Farm

- 14 x Magma EB16 PCIe expansion chassis
- Each EB16 contains 16 x GTX750Ti
- One or two EB16 per Dell 720 Host
 - 4 x Intel E5-2670
 - 2 x PCIe Expansion Cards
- Low power, low heat solution
- 314 GPU SP TFLOPs for \$100k



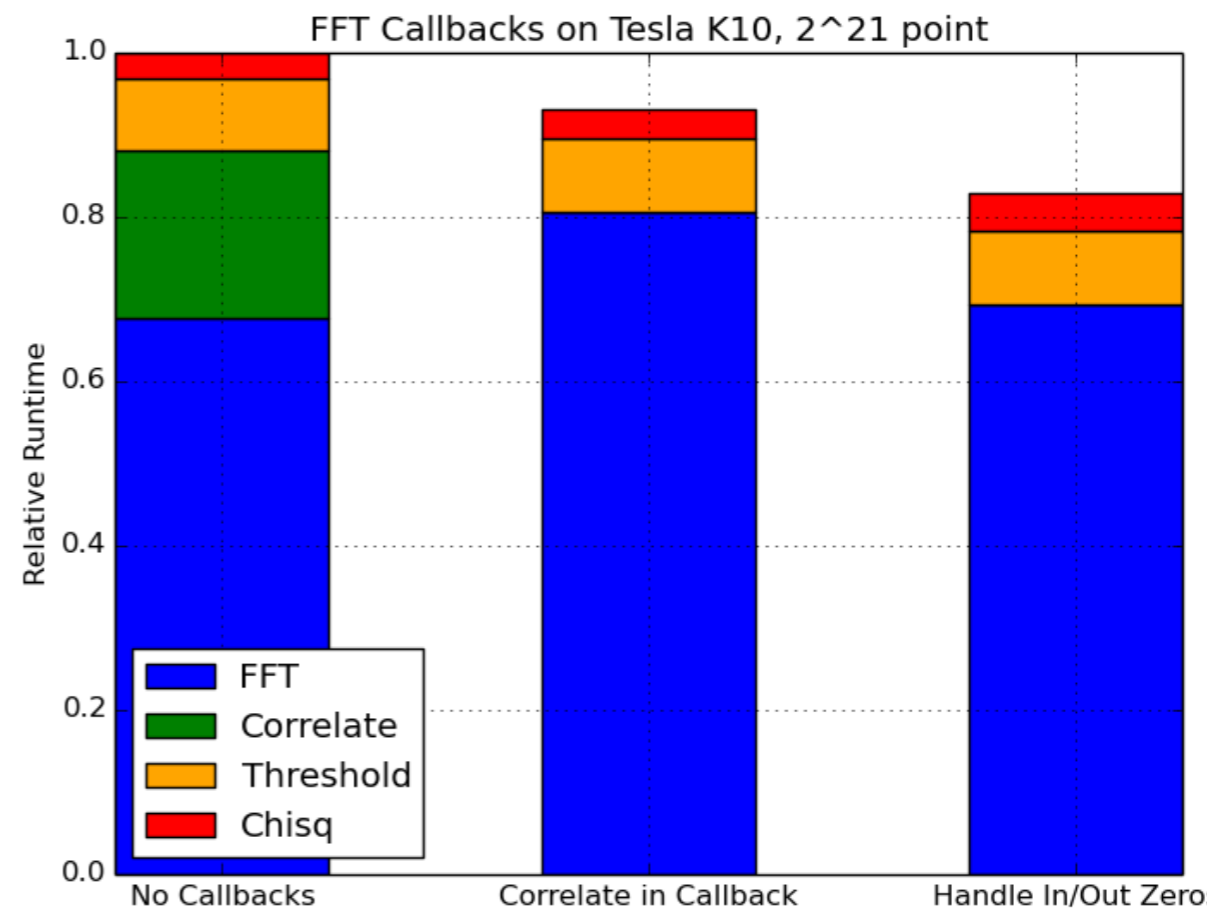
LIGO GPU Benchmarking

- The three kernels that dominate the inner loop are all GPU memory to compute memory bandwidth bound (on the GPU itself)
- Template generation, correlate, and event finding are currently implemented as separate kernels
- The cuFFT algorithm uses several kernels for each radix of the FFT and so triggers memory transfers during the FFT



LIGO GPU Optimization Plans

- Reduce number of kernel calls by using CUDA callbacks
- Prototype code has developed to fuse correlate and FFT into a single kernel call
- Increased speed by almost 20%
- Needs further development in collaboration with NVIDIA



Plans for O2

- 1500 GTX 750 GPU cards deployed on atlas.aei.uni-hannover.de
- 244 GTX 750 Ti GPU cards deployed on hercules.syr.edu
- Together these are equivalent to ~ 45,000 Sandybridge cores
- Initial testing shows that GPU and CPU codes produce consistent results for single triggers (up to numerical precision)
- Re-run full O1 search on GPU cards and compare results, if successful use GPUs for production searches later in O2.

Computing Challenges

- #1: Deploying / running optimized code to heterogenous computing environments. Interesting!
 - Dedicated internal, dedicated external, shared/allocated, opportunistic, commercial cloud, etc.
 - Many x86_64 CPU instruction sets, GPUs, MICs, KNL, etc.
- Properly-optimized executables can be up to factors-of-many faster than lowest-common-denominator executables.
- What we **don't** want to do: leave available resources un-utilized **or** run lowest-common-denominator code on newer platforms.
- Approaches:
 - Partition resources in advance, submit “right” code to each. (Eager planning.)
 - Pre-deploy “right” code to each resource in advance.
 - “Thin” dynamic/lazy payloads. (How late? What layer/s in the stack has the intelligence and participate identifying/retrieving the right code?)
 - “Fat” payloads, laziest possible determination.
 - Other?
- For this problem, I'm not sure it matters whether the payload is an executable, a “bundle”, a container, or a VM. Same basic problem.
- Has anyone solved it in a production scientific environment with many semi-autonomous users running existing codes, and many semi-autonomous heterogenous computing providers?

Computing Challenges

- #2: Usage accounting on heterogeneous computing environments. Boring!
 - Dedicated internal, dedicated external, shared/allocated, opportunistic, commercial cloud, etc.
 - Many x86_64 CPU instruction sets, GPUs, MICs, KNL, etc.
 - Different metrics: physical cores, generic core-hours, SUs, watts.
- You can account on the front end (submit-side) or the back end (execute-side) in between (grid middleware), or somehow try to combine/harmonize more than one.
- What we **don't** want to do:
 - Fail to account for resources used.
 - Fail to normalize CPU core-hours by relative performance. (But what's your benchmark?)
 - Double-count resources.
- Right now LIGO uses a combination of automated accounting and painstaking human merge. (Guess which human ultimately has to make sure it's right?)
- Can anyone help without invoking [xkcd #927](#)?

Computing Challenges

- #3: Deciding what & when to optimize and when not to. Interesting!
- This is tricky for a single developer and code. *Very* tricky for an entire collaboration and deep stacks of pipelines, middleware, infrastructure, and human processes.
- Determining the crossover point between the cost of human effort and the potential computation savings is key. **As much an art as a science.**
- We need to consider the (very rough) cost of labor, the (very rough) cost of a CPU hour, the potential gains from an optimization target, the likelihood of success or failure, and prioritize our efforts accordingly. Lots of human factors:
 - How competent are the developers? Is the search chair supportive?
 - How much of a pleasure (or pain in the neck) are they to work with? (Perverse incentives!)
 - How well does the optimization problem fit the skill sets of the people I can deploy?
 - How embarrassing would leaving this unoptimized be if our funding agency reviewed it?
 - How important is the scientific activity that would benefit?
 - How likely is the optimization to be long-lived (vs eclipsed by new technology or science)? I.e., how many years of payback for the up-front human investment?

Computing Challenges

- #4 GPUs: Interesting!
 - Accounting
 - Scheduling
 - 2 (?) hardware models: low-density vs high-density
 - Very different economics & scheduling.
 - Porting & maintaining codes
 - Generalization to other parallel architectures (e.g., KNL)?

Computing Challenges

- I have more, but I'm out of time!
- Find me during the break and I can talk your ear off.
- Thank you!

Acknowledgement

- H2020-Astronomy ESFRI and Research Infrastructure Cluster (Grant Agreement number: 653477).



Extra Slides

Science Priorities

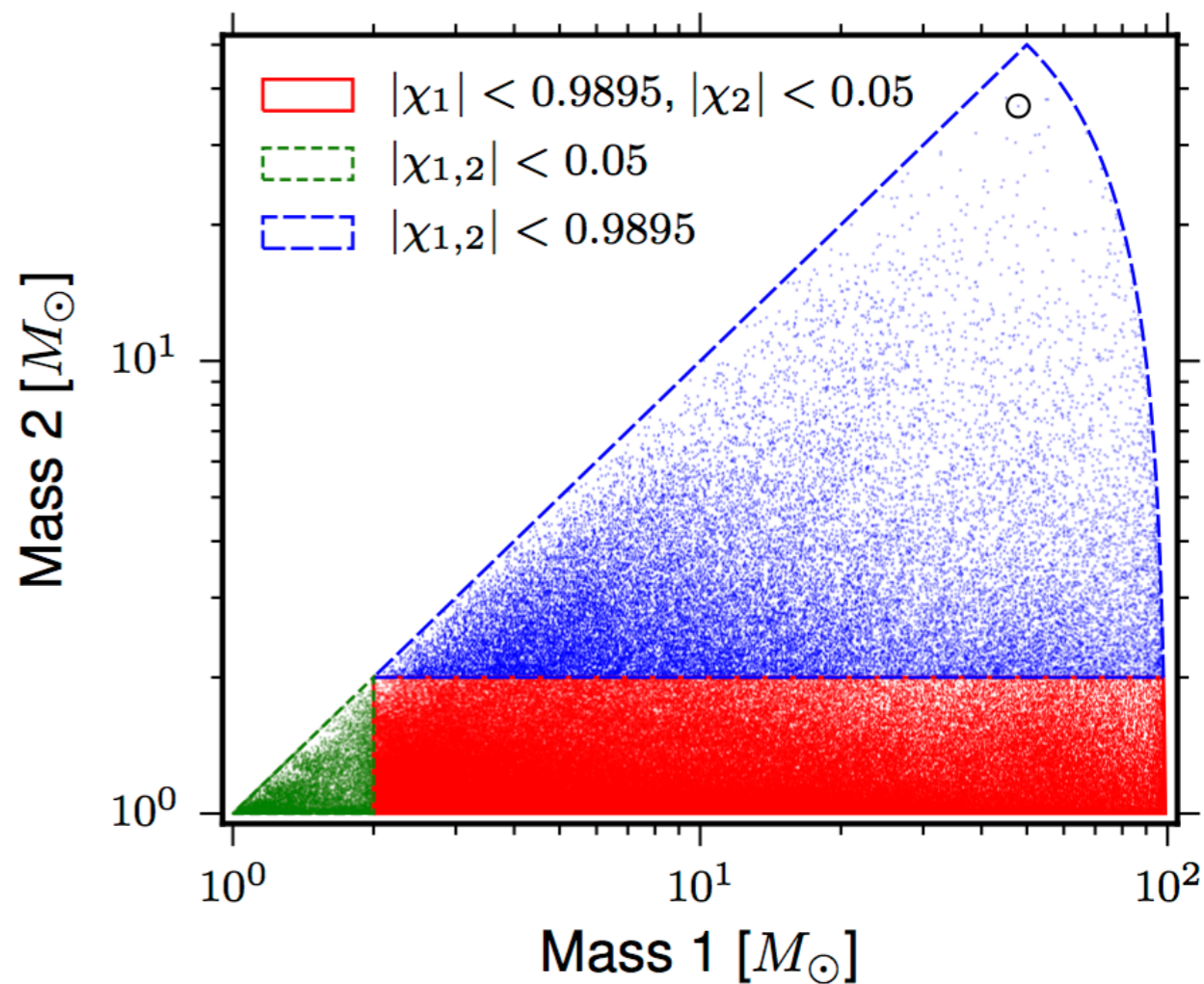
- Highest priority: searches most likely to make detections or yield significant astrophysical results;
- High priority: promising extensions of the highest priority goals that explore larger regions of parameter space or can further the science potential of LIGO and Virgo;
- Additional priority: sources with low detection probability but high scientific payoff.

	Burst	CBC	CW	SGWB
Highest priority	All-sky search for generic GW transients, in low latency for EM followup and deep, offline for 4σ detection confidence	Detecting the coalescence of neutron star and black hole binaries and measuring their parameters	All-sky search for isolated neutron stars, both as a <i>quick-look</i> on owned resources and as a deep/broad search on Einstein@Home	Directional search for stochastic GW background
	Parameter estimation for the astrophysical interpretation of detected burst events	Characterizing the astrophysical distribution of compact binaries	Targeted search for high value, known pulsars	Isotropic search for stochastic GW background
	Search for GW bursts triggered by outstanding GRB alerts	Responding to exceptional CBC detections	Directed searches for most promising isolated stars (Cas A, Vela Jr etc.)	Constraints of a detected background of astrophysical origin with long transients
	Searches triggered by outstanding astrophysical events (a galactic supernova, neutron star transients, an exceptional high energy neutrino alert)	Multi-messenger astronomy with compact binaries	Directed searches for X-ray binaries SCO-X1 and J1751-305	
	Search for cosmic string kinks and cusps	Searching for CBC-GRB coincidences Testing General Relativity with Compact Binaries		
High priority	Searches triggered by high energy neutrinos, extragalactic supernovae, and GRB observations	All sky search for spinning binary neutron star systems (deep and low latency)	Targeted search for other known pulsars	Long transient follow up of CBC and burst candidates
	Burst search for intermediate mass ratio and eccentric black hole binary systems	Matched filtered search for intermediate mass black hole binary systems	Directed searches for other isolated stars and X-ray binaries	
	All-sky search for long bursts of > 10 s duration			
Additional priority	GRB-triggered search for long-duration bursts and plateaus	Exploring effects of detector noise on parameter estimation	All sky search for isolated stars (alternative approaches)	
	Hypermassive neutron star followup	Searching for sub-solar mass CBC signals	All-sky search for binaries	
	Burst searches triggered by radio transients and by SGR/SGR-QPO	Developing searches for CBC signals with generic spins	Spotlight deep sky-patch search **	
	Burst tests of alternative gravity theories **		Search for Supernova post birth signals **	
			Search for continuous wave transients **	

Table 2 in LIGO Document T1600115

LIGO Impact of Template Waveforms

- Two different waveform families used in O1 depending on target source:
 - Post-Newtonian (PN)
 - Effective One Body (EOB)
- Very different computationally
 - PN is expressed as a frequency-domain polynomial
 - EOB is numerically integrated in the time domain



GPU Template Implementation

- Currently no GPU implementation of the Effective One Body waveforms used for binary black hole searches
- Grad student project underway to create a GPU version of SEOBNRv2 Reduced Order Model waveform used in O1
 - Core of algorithm is cubic b-spline interpolation, CUDA implementations available
- Investigating low-cost waveform generation techniques for a that can be template-family agnostic
 - Important since template families can change in response to science needs