Sustainability Challenges:

What Does It Take to Keep PAPI Instrumental for the HPC Community?

Heike Jagode

Anthony Danalis, Jack Dongarra

2019 Collegeville Workshop on
Sustainable Scientific Software (CW3S19)

July 22-24, 2019

INNOVATIVE
COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

PAPI

- Library that provides a **consistent interface** (and methodology) for hardware performance counters, found across the system:
  i. e., CPUs, GPUs, on-/off-chip Memory, Interconnects, I/O system, File System, Energy/Power, etc.
- PAPI enables software engineers to see, in near real time, the relation between **SW performance** and **HW events across the entire compute system**

PAPI

- Library that provides a **consistent interface** (and methodology) for hardware performance counters, found across the system:
  i. e., CPUs, GPUs, on-/off-chip Memory, Interconnects, I/O system, File System, Energy/Power, etc.
- PAPI enables software engineers to see, in near real time, the relation between
  SW performance and **HW events across the entire compute system**

SUPPORTED ARCHITECTURES:

- AMD **up to Zeppelin Zen**
- ARM Cortex A8, A9, A15, ARM64
- IBM Blue Gene Series
- IBM Power Series, **PCP for POWER9-uncore**
- Intel SandyIIvy Bridge, Haswell, Broadwell, Skylake, **Kabylake**, **Cascadelake,** KNC, KNL, KNM

AMD
ARM
CRAY
THE SUPERCOMPUTER COMPANY
IBM
intel
NVIDIA

PAPI

- Library that provides a **consistent interface** (and methodology) for hardware performance counters, found across the system:
  i. e., CPUs, GPUs, on-/off-chip Memory, Interconnects, I/O system, File System, Energy/Power, etc.
- PAPI enables software engineers to see, in near real time, the relation between **SW performance** and **HW events across the entire compute system**

SUPPORTED ARCHITECTURES:

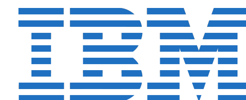
- AMD **up to Zeppelin Zen**,
- AMD **GPUs Vega**
- ARM Cortex A8, A9, A15, ARM64
- CRAY: Gemini and Aries interconnects, power/energy
- IBM Blue Gene Series, Q: 5D-Torus, I/O system, EMON power/energy
- IBM Power Series, **PCP for POWER9-uncore**
- Intel SandyIIvy Bridge, Haswell, Broadwell, Skylake, **Kabylake**, **Cascadelake,** KNC, KNL, KNM
- InfiniBand
- Lustre FS
- NVIDIA Tesla, Kepler, Maxwell, Pascal, **Volta**: support for multiple GPUs
- **NVIDIA: support for NVLink**

AMD ARM CRAY THE SUPERCOMPUTER COMPANY IBM intel NVIDIA

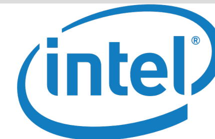
PAPI

- Library that provides a **consistent interface** (and methodology) for hardware performance counters, found across the system: i. e., CPUs, GPUs, on-/off-chip Memory, Interconnects, I/O system, File System, Energy/Power, etc.
- PAPI enables software engineers to see, in near real time, the relation between **SW performance** and **HW events across the entire compute system**

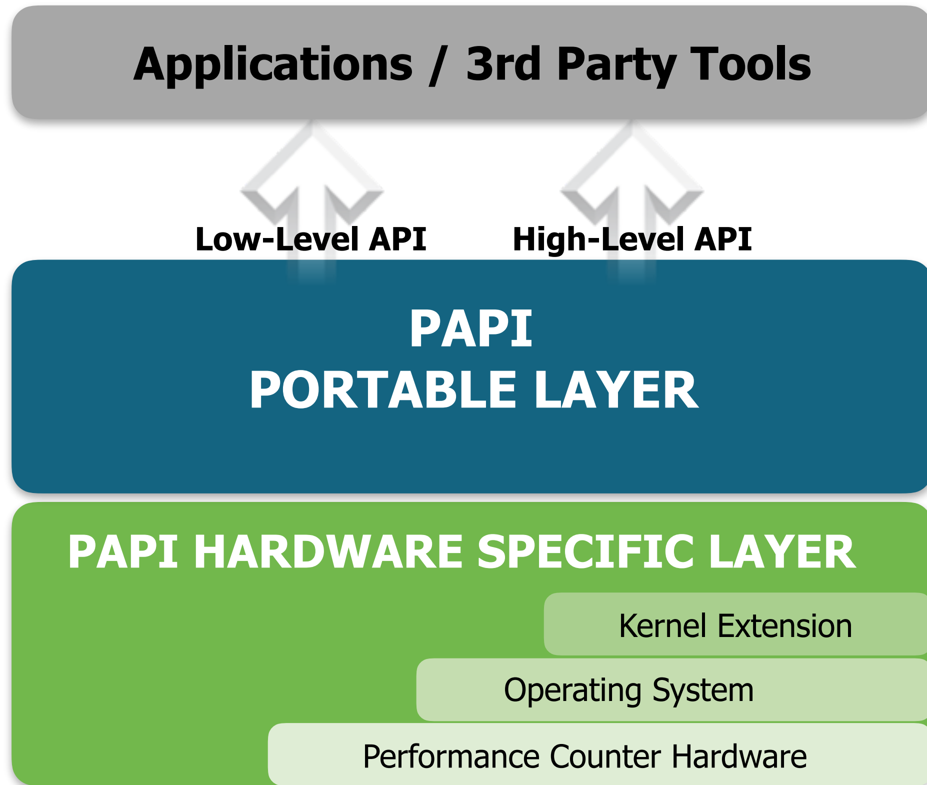
SUPPORTED ARCHITECTURES:



- AMD **up to Zeppelin Zen**, **power for Fam17h**
- AMD **GPUs Vega**, **power, temperature, fan**
- ARM Cortex A8, A9, A15, ARM64
- CRAY: Gemini and Aries interconnects, power/energy
- IBM Blue Gene Series, Q: 5D-Torus, I/O system, EMON power/energy
- IBM Power Series, **PCP for POWER9-uncore**
- Intel SandyIIvy Bridge, Haswell, Broadwell, Skylake, **Kabylake**, **Cascadelake,** KNC, KNL, KNM
- Intel RAPL (power/energy), **power capping**
- InfiniBand
- Lustre FS
- NVIDIA Tesla, Kepler, Maxwell, Pascal, **Volta**: support for multiple GPUs
- **NVIDIA: support for NVLink**
- NVIDIA NVML (power/energy); **power capping**
- Virtual Environments: VMware, KVM



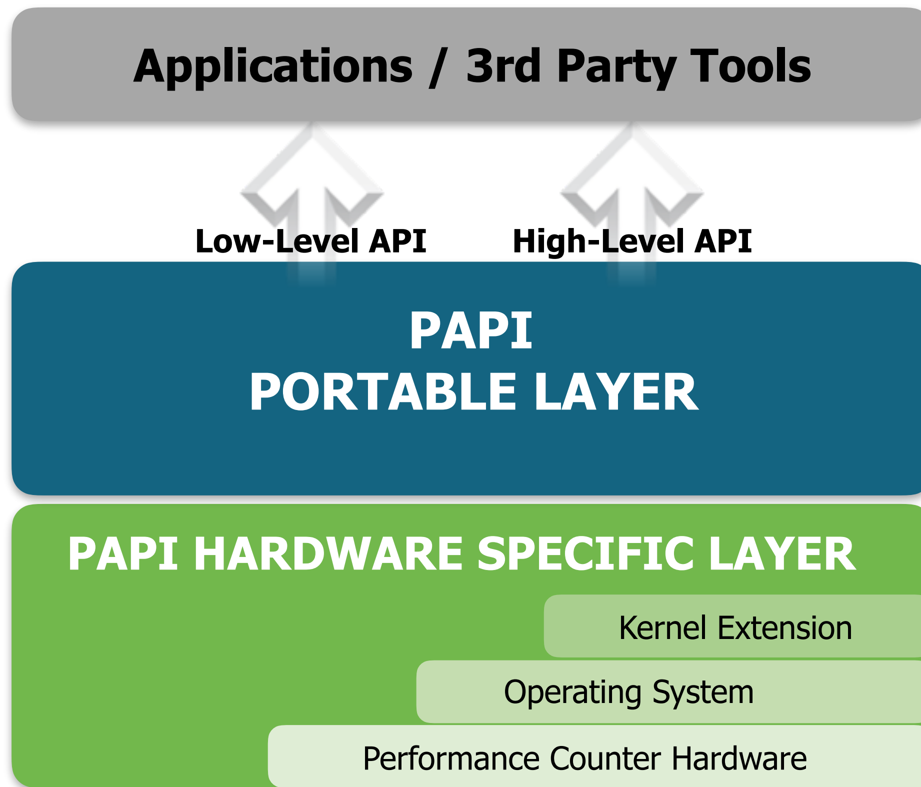
PAPI Framework: 1999 - 2009



PAPI's original job:

Address the **problem of accessing hardware counters,** found on a diverse collection of modern microprocessors, **in a portable manner.**

PAPI Framework: 1999 - 2009



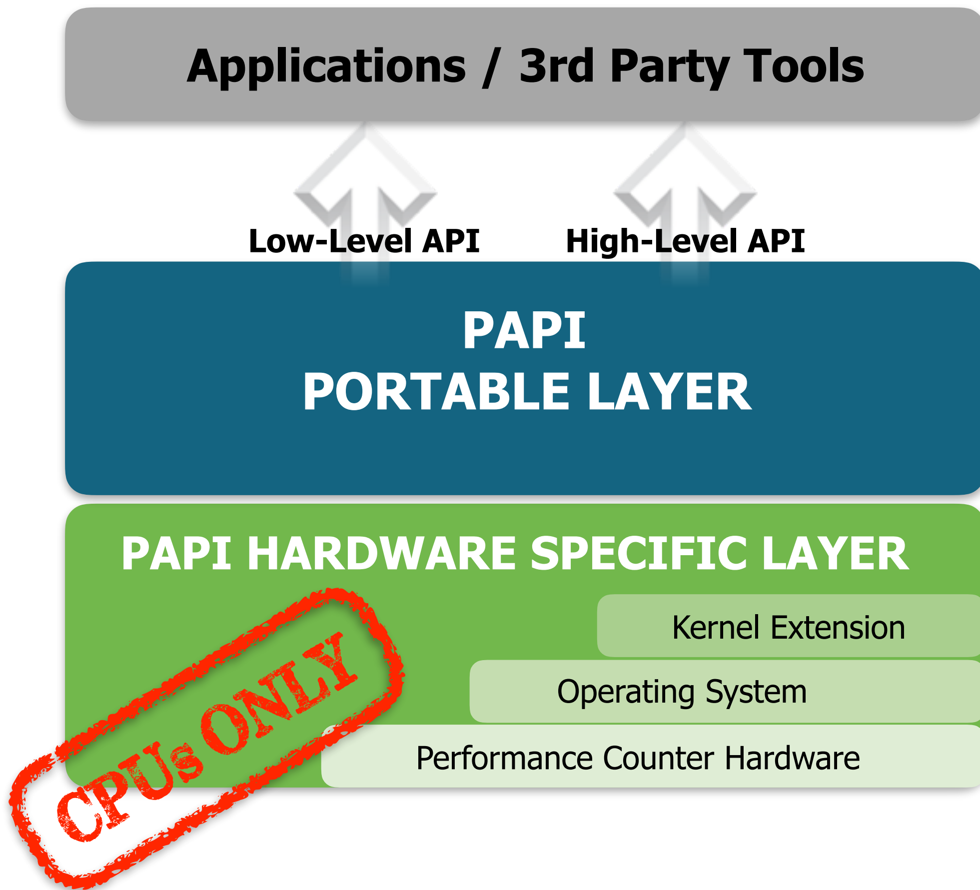
PAPI's original job:

Address the **problem of accessing hardware counters,** found on a diverse collection of modern microprocessors, **in a portable manner.**

Why?

- Too many different interfaces from different CPU vendors
- Interfaces poorly documented
- Performance counters poorly documented (or not documented)
- Number of counters (offered by vendors) has vastly increased over the years, and so has their complexity
- No standardized way to access these counters

PAPI Framework: 1999 - 2009



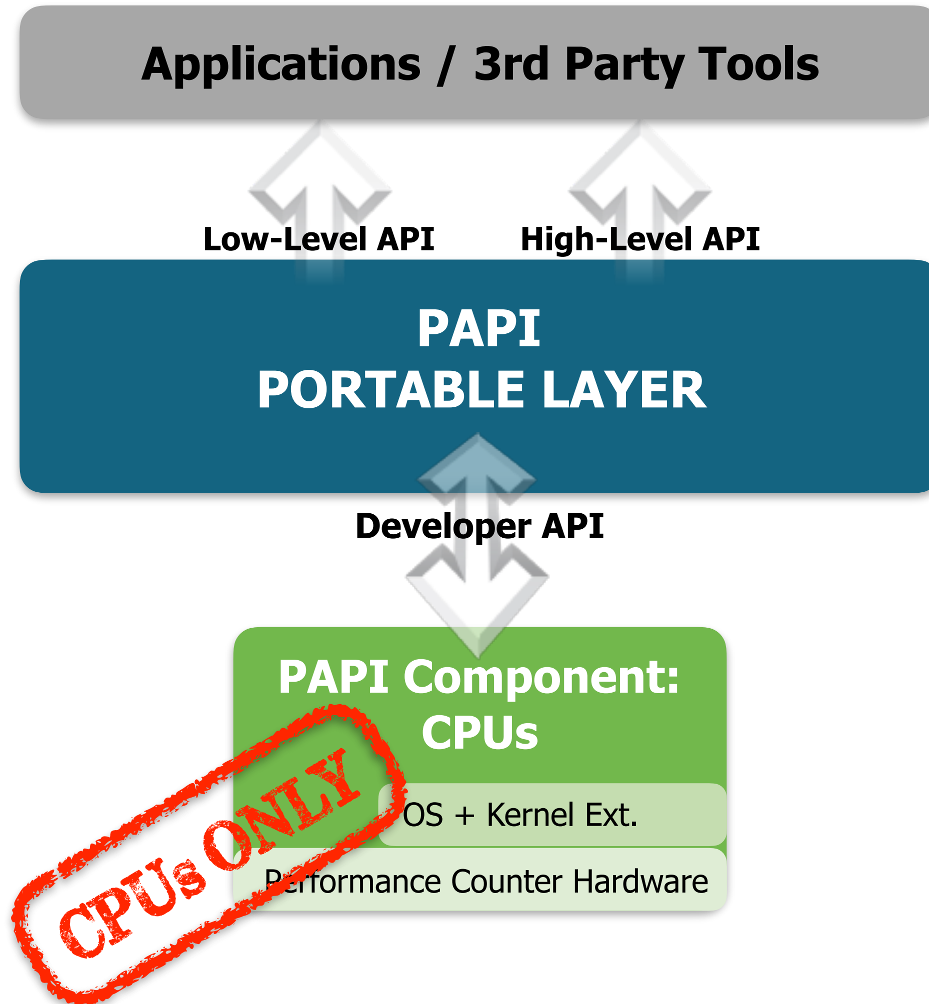
PAPI's original job:

Address the **problem of accessing hardware counters,** found on a diverse collection of modern microprocessors, **in a portable manner.**

Why?

- Too many different interfaces from different CPU vendors
- Interfaces poorly documented
- Performance counters poorly documented (or not documented)
- Number of counters (offered by vendors) has vastly increased over the years, and so has their complexity
- No standardized way to access these counters

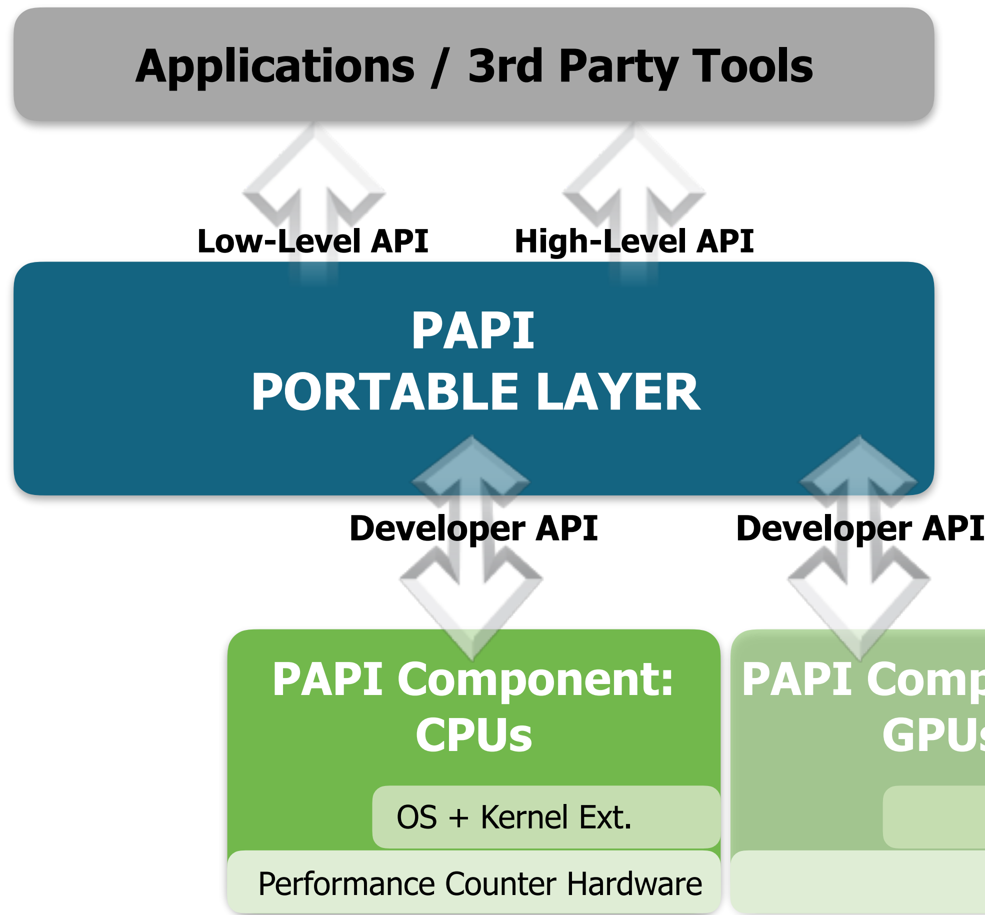
PAPI Framework: 2009



Challenge:

Tools to measure application performance **in increasingly complex systems MUST** also **increase the richness of their measurements** to provide insights into the increasingly intricate ways in which SW and HW interact!

PAPI Framework: 2009 - 2018

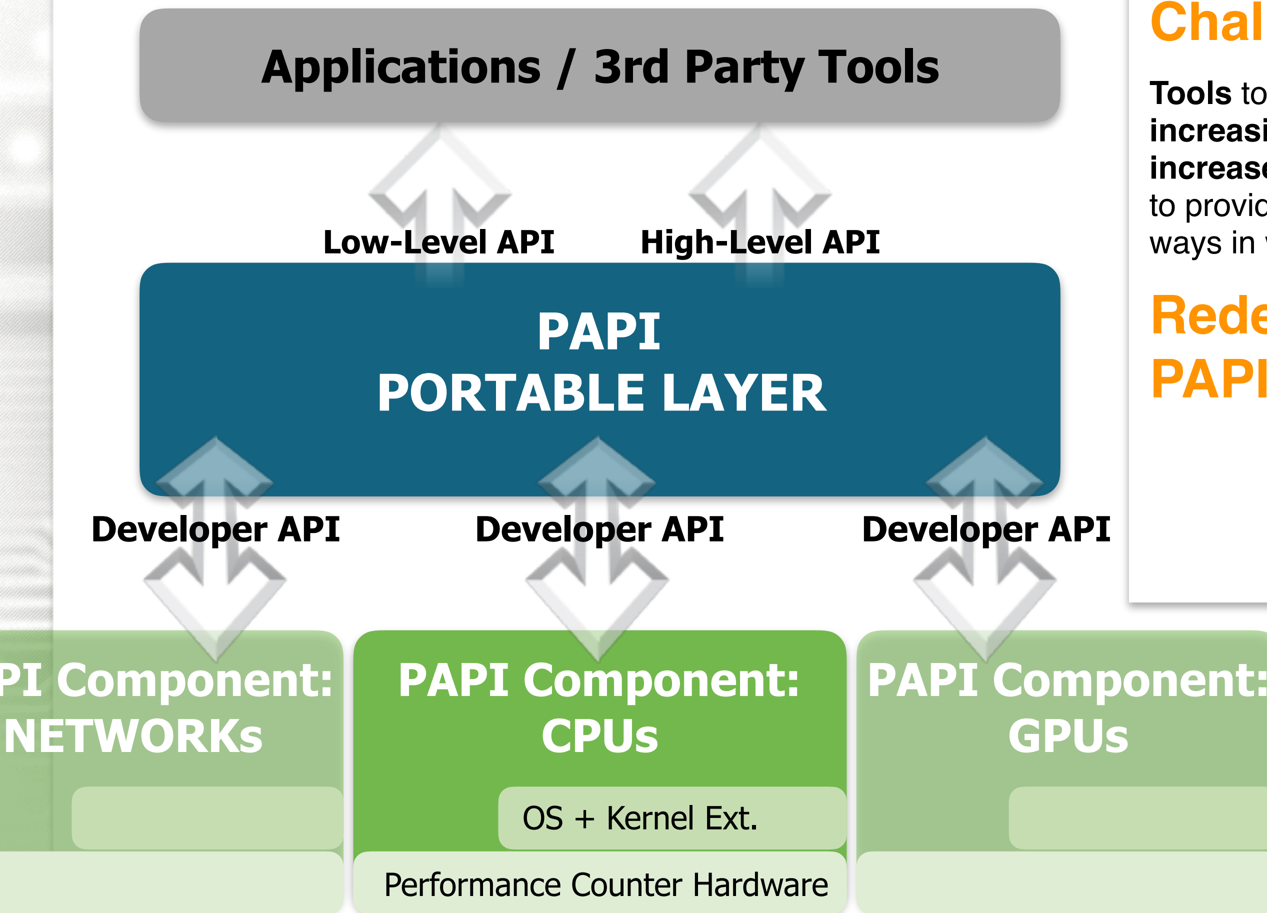


Challenge:

Tools to measure application performance **in increasingly complex systems MUST** also **increase the richness of their measurements** to provide insights into the increasingly intricate ways in which SW and HW interact!

Redesign of PAPI to PAPI-Components:

PAPI Framework: 2009 - 2018

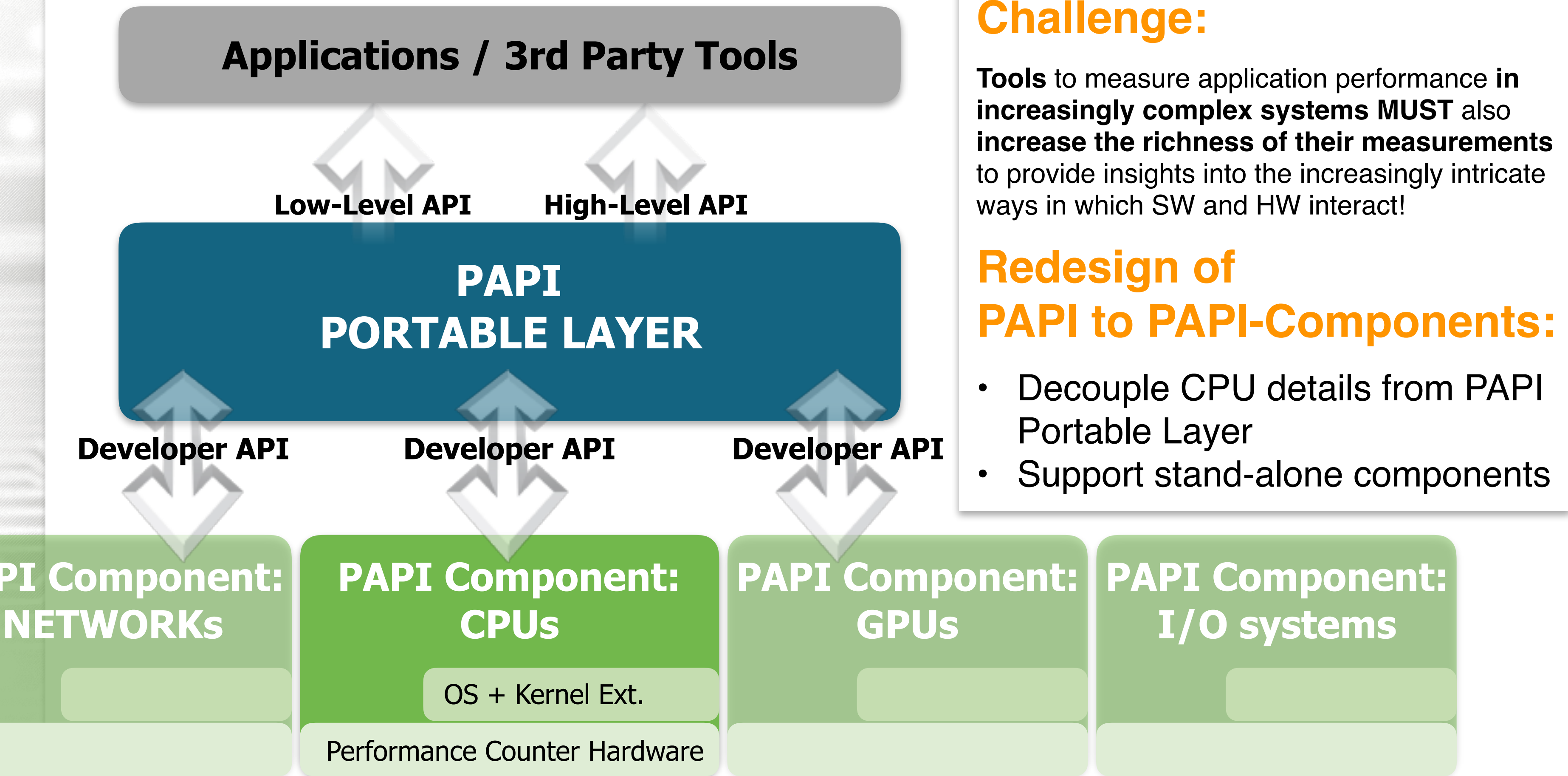


Challenge:

Tools to measure application performance **in increasingly complex systems MUST** also **increase the richness of their measurements** to provide insights into the increasingly intricate ways in which SW and HW interact!

Redesign of PAPI to PAPI-Components:

PAPI Framework: 2009 - 2018



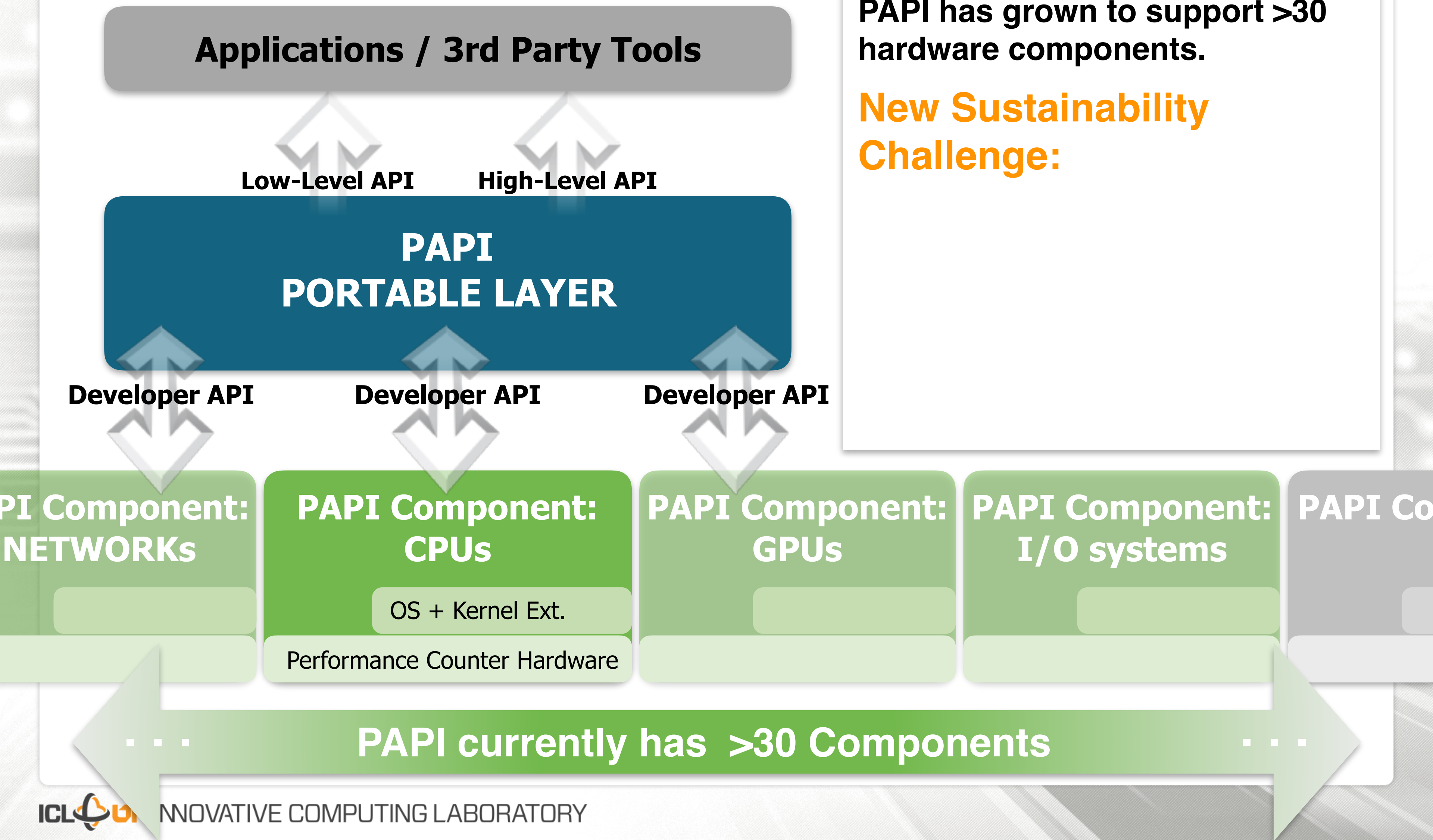
Challenge:

Tools to measure application performance **in increasingly complex systems MUST** also **increase the richness of their measurements** to provide insights into the increasingly intricate ways in which SW and HW interact!

Redesign of PAPI to PAPI-Components:

- Decouple CPU details from PAPI Portable Layer
- Support stand-alone components

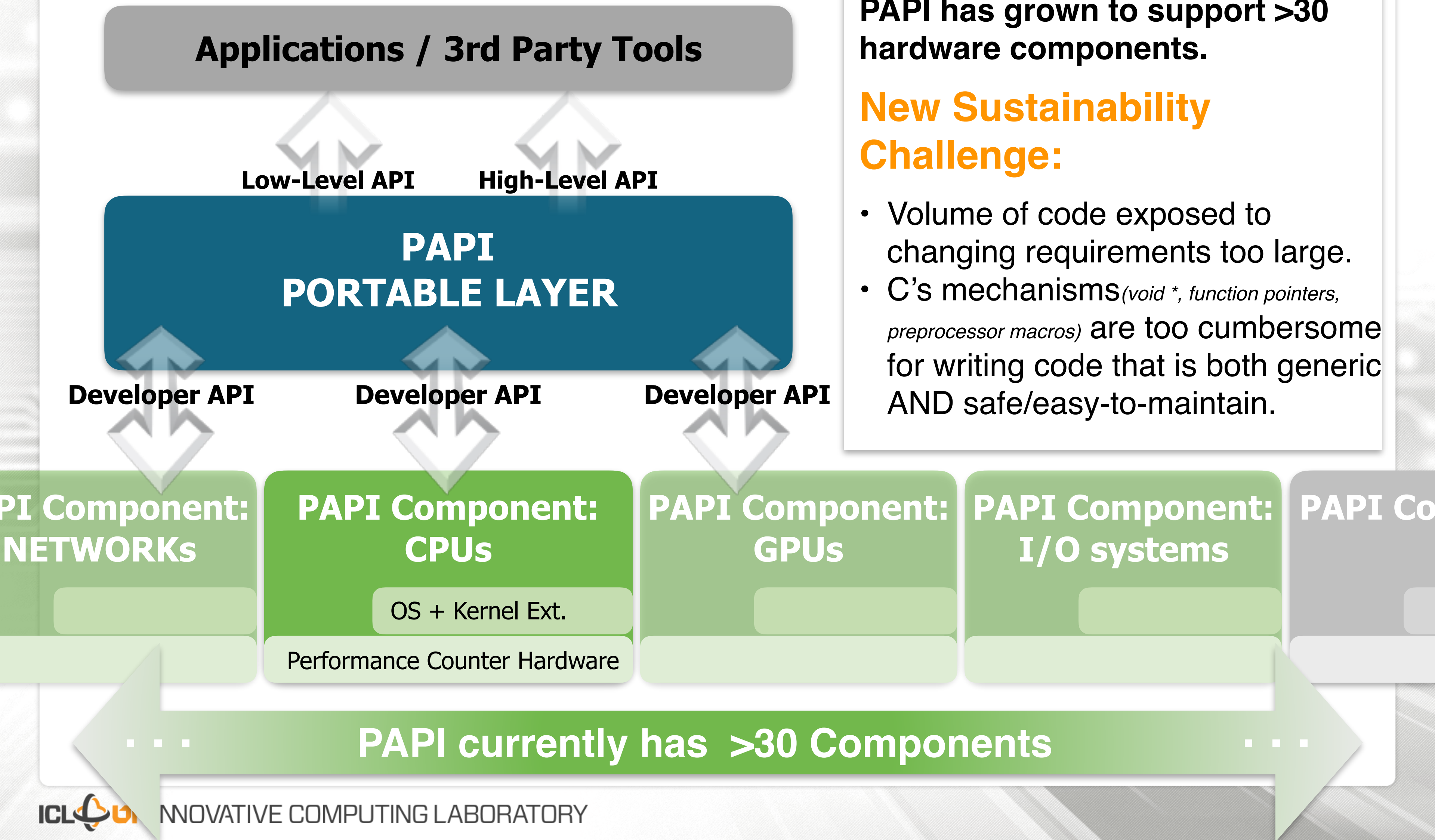
PAPI Framework: 2009 - 2018



PAPI has grown to support >30 hardware components.

New Sustainability Challenge:

PAPI Framework: 2009 - 2018

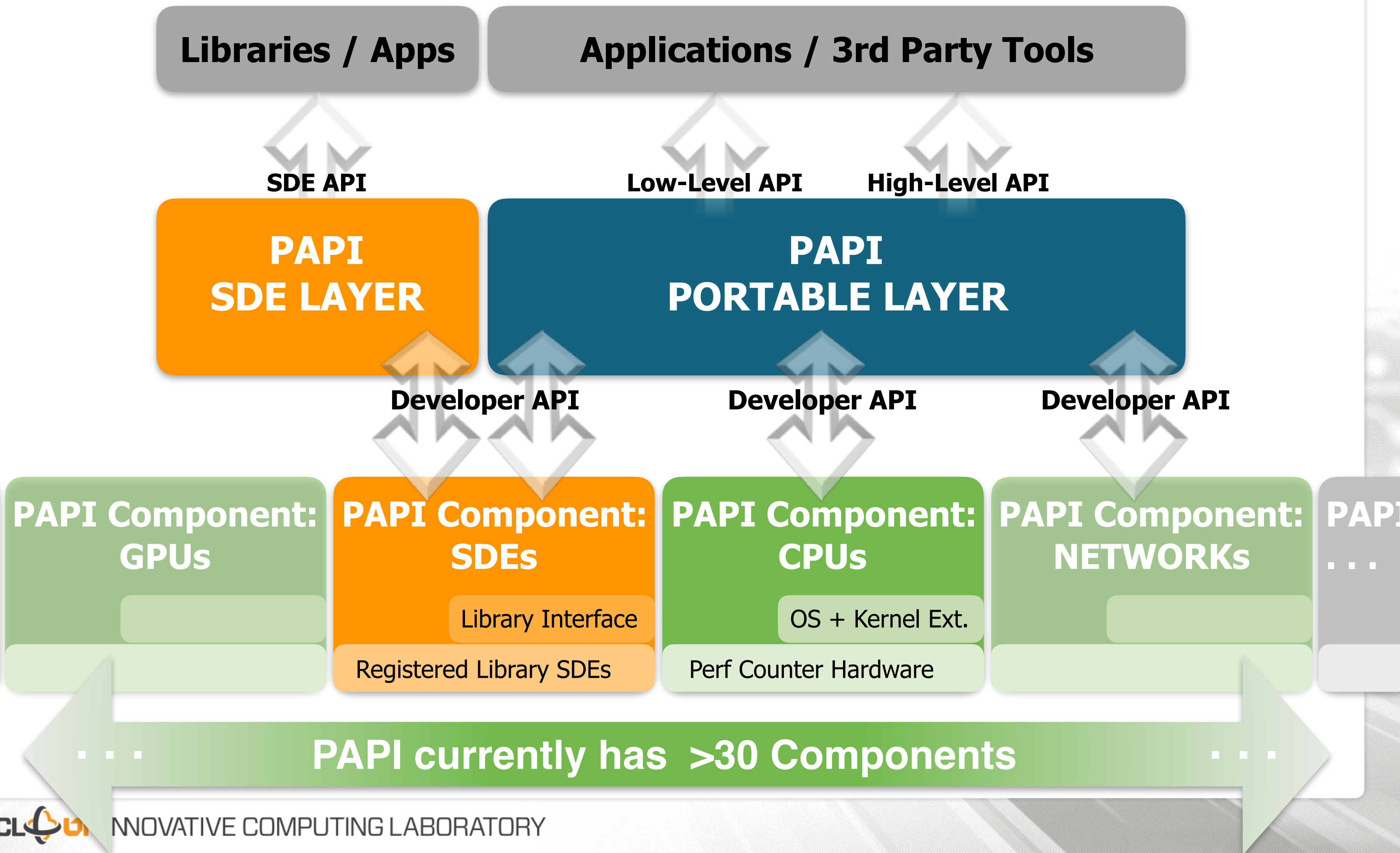


PAPI has grown to support >30 hardware components.

New Sustainability Challenge:

- Volume of code exposed to changing requirements too large.
- C's mechanisms *(void *, function pointers, preprocessor macros)* are too cumbersome for writing code that is both generic AND safe/easy-to-maintain.

PAPI Framework: 2018 - present



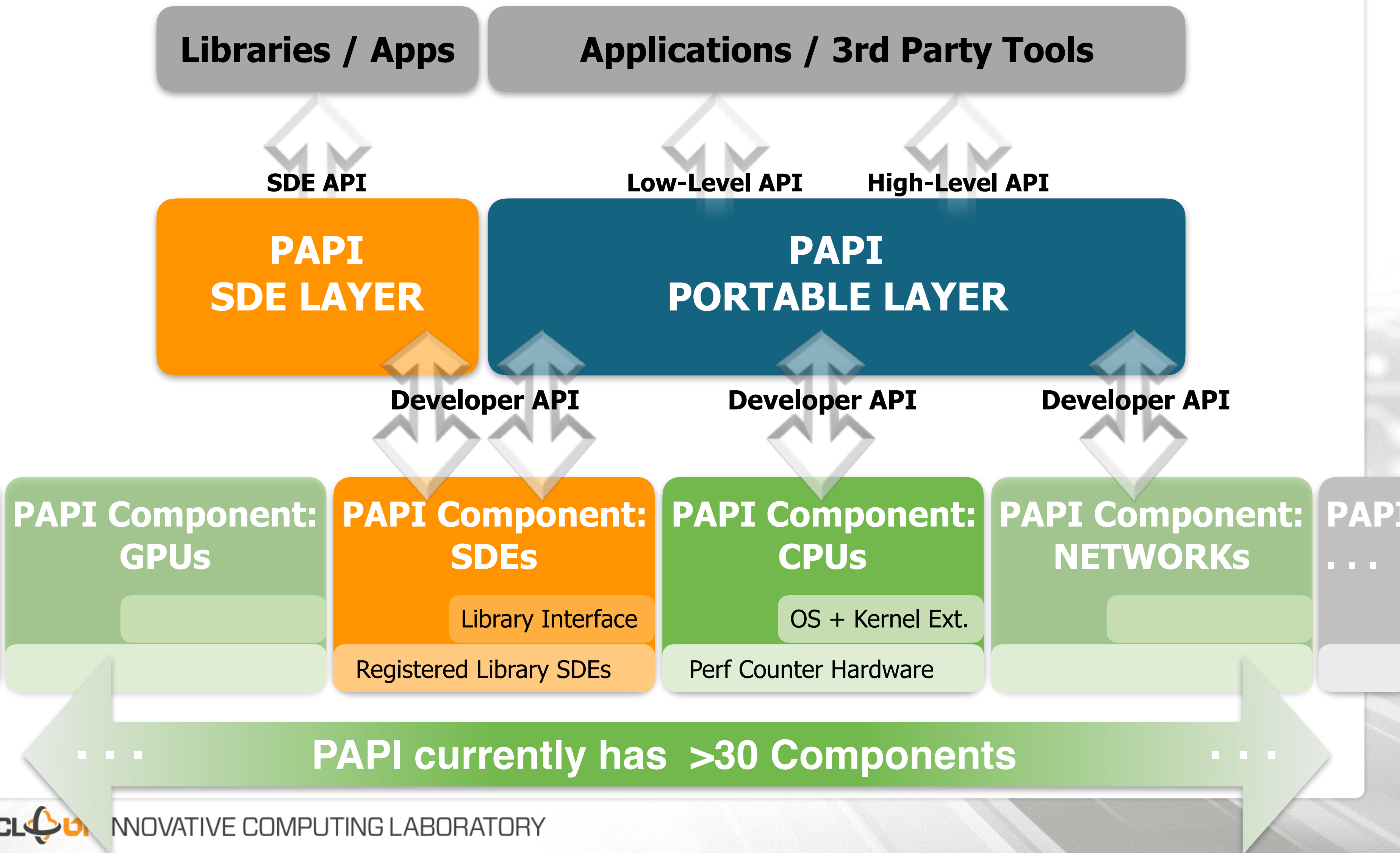
PAPI Software-defined Event (SDE) Support

GOAL Offer support for software-defined events (SDE) to extend PAPI's role as a standardizing layer for monitoring performance counters.

VISION Enable HPC software layers to expose SDEs that performance analysts can use to form a complete picture of the entire application performance.

BENEFIT HPC application scientists will be able to better understand the interaction of the different application layers, and the interaction with external libraries and runtimes.

PAPI Framework: 2018 - present



PAPI SDE API

```
void *papi_sde_init(const char *lib_name);
```

Initializes internal data structures and **returns an opaque handle** that must be passed to all subsequent calls to PAPI SDE functions.

`lib_name` is a string containing the name of the library.

```
int papi_sde_register_counter(void *handle, const char *event_name,
                              int mode, int type, void *counter);
```

Must be called for every program variable that the library wishes to register as an event.

`handle` is the opaque handle returned by `papi_sde_init()`.
`event_name` is a string containing the name of the event being registered.
`mode` is an integer declaring whether a counter is read-only or read-write.
`type` is an enumeration of the type of the event.
`counter` is a pointer to the actual variable that serves as the counter for this event.

```
typedef long long (*papi_sde_fptr_t)(void *);
int papi_sde_register_fp_counter(void *handle, const char *event_name,
           int mode, int type, papi_sde_fptr_t fp_counter, void *param);
```

Registers a function pointer to an accessor function provided by the library. Enables the user to export an event the value of which does not map to the value of a single program variable inside the library.

`fp_counter` is a pointer to the accessor function with return type "`long long int`".
`param` is an opaque object that the library passes to PAPI, and PAPI passes it as a parameter to the accessor function.

```
int papi_sde_unregister_counter(void *handle, const char *event_name);
```

Can be called to unregister an event counter. Useful for implementing transient events.

```
int papi_sde_describe_counter(void *handle, const char *event_name,
                              const char *event_description);
```

Provides a description of the event which will be displayed by utilities such as papi_native_avail.

`event_description` is a string containing the description of the event.

```
int papi_sde_add_counter_to_group(void *handle, const char *event_name,
                           const char *group_name, uint32_t group_flags);
```

Adds a counter to a group so that logical groups can be formed out of multiple related event counters. Groups are first class citizens and can be recursively added to other groups. A group is automatically created the first time a counter is added to it.

`group_name` is a string containing the name of the group.
`group_flags` specifies if the group should report the `sum`, the `min`, or the `max` of the counters it contains.

```
int papi_sde_create_counter(void *handle, const char *event_name,
                            int type, void *counter_handle);
```

Creates a counter whose memory is managed by PAPI (instead of the library)

`handle` is the opaque handle returned by `papi_sde_init()`.
`event_name` is a string containing the name of the event being registered.
`type` is an enumeration of the type of the event.
`counter_handle` is a opaque handle that can be used to access the created counter

```
int papi_sde_inc_counter(void *counter_handle, long long increment);
```

Creates a counter whose memory is managed by PAPI (instead of the library)

`counter_handle` is the opaque handle returned by `papi_sde_create_counter()`.
`increment` is the value to be added to the counter.

```
int papi_sde_create_recorder(void *handle, const char *event_name,
                             size_t typesize, void *record_handle);
```

Creates a counter which can record (log) a series of values. The memory of the recorder is handled internaly by PAPI.

`handle` is the opaque handle returned by `papi_sde_init()`.
`event_name` is a string containing the name of the event being registered.
`typesize` is the size of each element (to be recorded) in bytes .
`record_handle` is a opaque handle that can be used to access the created recorder.

```
int papi_sde_record(void *record_handle, size_t typesize, void *value);
```

Records an element

`record_handle` is the opaque handle returned by `papi_sde_create_recorder()`.
`typesize` is the size of the new element in bytes.
`value` is a pointer to the new element.

PAPI SDE API

```
void *papi_sde_init(const char *lib_name);
```

Initializes internal data structures and **returns an opaque handle** that must be passed to all subsequent calls to PAPI SDE functions.

`lib_name` is a string containing the name of the library.

```
int papi_sde_register_counter(void *handle, const char *event_name,
                              int mode, int type, void *counter);
```

Must be called for every program variable that the library wishes to register as an event.

`handle` is the opaque handle returned by `papi_sde_init()`.
`event_name` is a string containing the name of the event being registered.
`mode` is an integer declaring whether a counter is read-only or read-write.
`type` is an enumeration of the type of the event.
`counter` is a pointer to the actual variable that serves as the counter for this event.

```
typedef long long (*papi_sde_fptr_t)(void *);
int papi_sde_register_fp_counter(void *handle, const char *event_name,
         int mode, int type, papi_sde_fptr_t fp_counter, void *param);
```

Registers a function pointer to an accessor function provided by the library. Enables the user to export an event the value of which does not map to the value of a single program variable inside the library.

`fp_counter` is a pointer to the accessor function with return type "`long long int`".
`param` is an opaque object that the library passes to PAPI, and PAPI passes it as a parameter to the accessor function.

```
int papi_sde_unregister_counter(void *handle, const char *event_name);
```

Can be called to unregister an event counter. Useful for implementing transient events.

```
int papi_sde_describe_counter(void *handle, const char *event_name,
                              const char *event_description);
```

Provides a description of the event which will be displayed by utilities such as papi_native_avail.

`event_description` is a string containing the description of the event.

```
int papi_sde_add_counter_to_group(void *handle, const char *event_name,
                                  const char *group_name, uint32_t group_flags);
```

Adds a counter to a group so that logical groups can be formed out of multiple related event counters. Groups are first class citizens and can be recursively added to other groups. A group is automatically created the first time a counter is added to it.

`group_name` is a string containing the name of the group.
`group_flags` specifies if the group should report the `sum`, the `min`, or the `max` of the counters it contains.

```
int papi_sde_create_counter(void *handle, const char *event_name,
                            int type, void *counter_handle);
```

Creates a counter whose memory is managed by PAPI (instead of the library)

`handle` is the opaque handle returned by `papi_sde_init()`.
`event_name` is a string containing the name of the event being registered.
`type` is an enumeration of the type of the event.
`counter_handle` is a opaque handle that can be used to access the created counter

```
int papi_sde_inc_counter(void *counter_handle, long long increment);
```

Creates a counter whose memory is managed by PAPI (instead of the library)

`counter_handle` is the opaque handle returned by `papi_sde_create_counter()`.
`increment` is the value to be added to the counter.

```
int papi_sde_create_recorder(void *handle, const char *event_name,
                             size_t typesize, void *record_handle);
```

Creates a counter which can record (log) a series of values. The memory of the recorder is handled internaly by PAPI.

`handle` is the opaque handle returned by `papi_sde_init()`.
`event_name` is a string containing the name of the event being registered.
`typesize` is the size of each element (to be recorded) in bytes .
`record_handle` is a opaque handle that can be used to access the created recorder.

```
int papi_sde_record(void *record_handle, size_t typesize, void *value);
```

Records an element

`record_handle` is the opaque handle returned by `papi_sde_create_recorder()`.
`typesize` is the size of the new element in bytes.
`value` is a pointer to the new element.

Advanced PAPI SDE API offers a variety of feature:

- **single events, recorders,**
- **function pointers,**
- **statistics,**
- **groups, …**

SDEs in MAGMA-SPARSE

Single-value SDEs: Simple register one counter via:

```
int papi_sde_register_counter( …, const char *event_name,…,
                                void *counter);
```

MAGMA performance metrics	Description
MAGMA::numiter	Number of iterations until convergence attained
MAGMA::SpmvCount	Number of sparse matrix-vector multiplications
MAGMA::InitialResidual	Initial residual
MAGMA::FinalResidual	Final residual
MAGMA::SolverRuntime	Total run-time of the solver

SDEs in MAGMA-SPARSE

Single-value SDEs: Simple register one counter via:

```
int papi_sde_register_counter( …, const char *event_name,…,
                                  void *counter);
```

MAGMA performance metrics	Description
MAGMA::numiter	Number of iterations until convergence attained
MAGMA::SpmvCount	Number of sparse matrix-vector multiplications
MAGMA::InitialResidual	Initial residual
MAGMA::FinalResidual	Final residual
MAGMA::SolverRuntime	Total run-time of the solver

Multi-value SDEs: Create a recorder to log multiple values for one SDE:

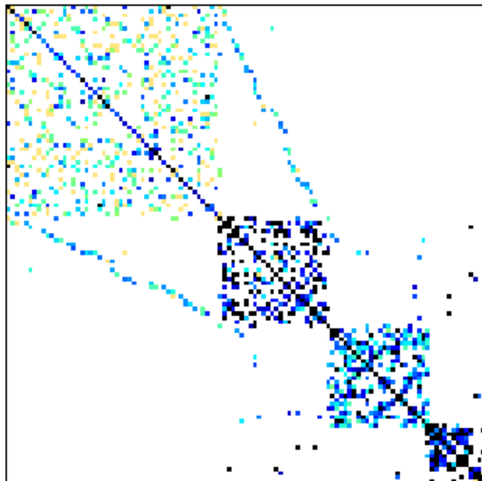
```
int papi_sde_create_recorder( …, const char *event_name,…,
                                 void *record_handle);
int papi_sde_record( void *record_handle,…, void *value);
```

MAGMA performance metrics	Description
MAGMA::IterativeResidual	Recorder of all residuals until convergence

SDEs in MAGMA-SPARSE: Power Network Problem

symmetric and positive-definite matrix

sde:::MAGMA::numiter:	555	•••	784	•••	880	•••
sde:::MAGMA::SpmvCount:	1110		1568		1760	
sde:::MAGMA::InitialResidual:	2.4019e+03		2.4019e+03		2.4019e+03	
sde:::MAGMA::FinalResidual:	6.4937e-10		5.8935e-10		7.2674e-10	
sde:::MAGMA::SolverRuntime:	9.5568e-02		1.1197e-01		1.7255e-01	



PAPI SDE Recorder: Residual per Iteration (662_bus: 662-by-662 with 2474 nonzeros)

Residual: 10^{6}, 10^{4}, 10^{2}, 10^{0}, 10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}

Iteration: 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000

BICG
BICGSTAB
CG
CGS
GMRES
IDR
QMR
TFQMR

Runtime of Solvers:
0.1312 sec
0.0956 sec
0.0404 sec
0.1120 sec
0.6497 sec
0.2999 sec
0.1725 sec
0.1910 sec

Matrix from: https://sparse.tamu.edu/HB/662_bus

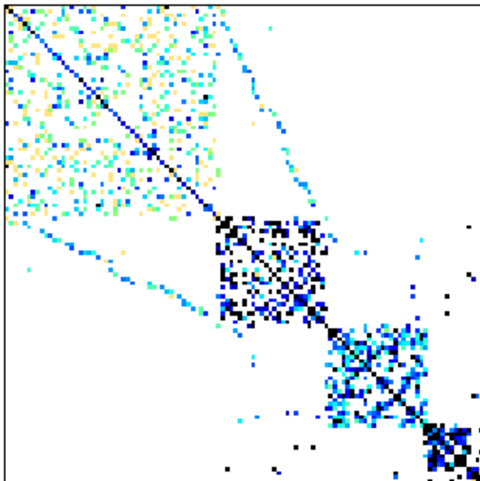
SDEs in MAGMA-SPARSE: Power Network Problem

symmetric and positive-definite matrix

sde:::MAGMA::numiter:	555	•••	784	•••	880	•••
sde:::MAGMA::SpmvCount:	1110		1568		1760	
sde:::MAGMA::InitialResidual:	2.4019e+03		2.4019e+03		2.4019e+03	
sde:::MAGMA::FinalResidual:	6.4937e-10		5.8935e-10		7.2674e-10	
sde:::MAGMA::SolverRuntime:	9.5568e-02		1.1197e-01		1.7255e-01	

IterativeResidual:CNT:

BICGSTAB = 555
CGS = 784
QMR = 880
••• •••



PAPI SDE Recorder: Residual per Iteration (662_bus: 662-by-662 with 2474 nonzeros)

Residual (10^{-10} to 10^{6}) vs. Iteration (0 to 1000)

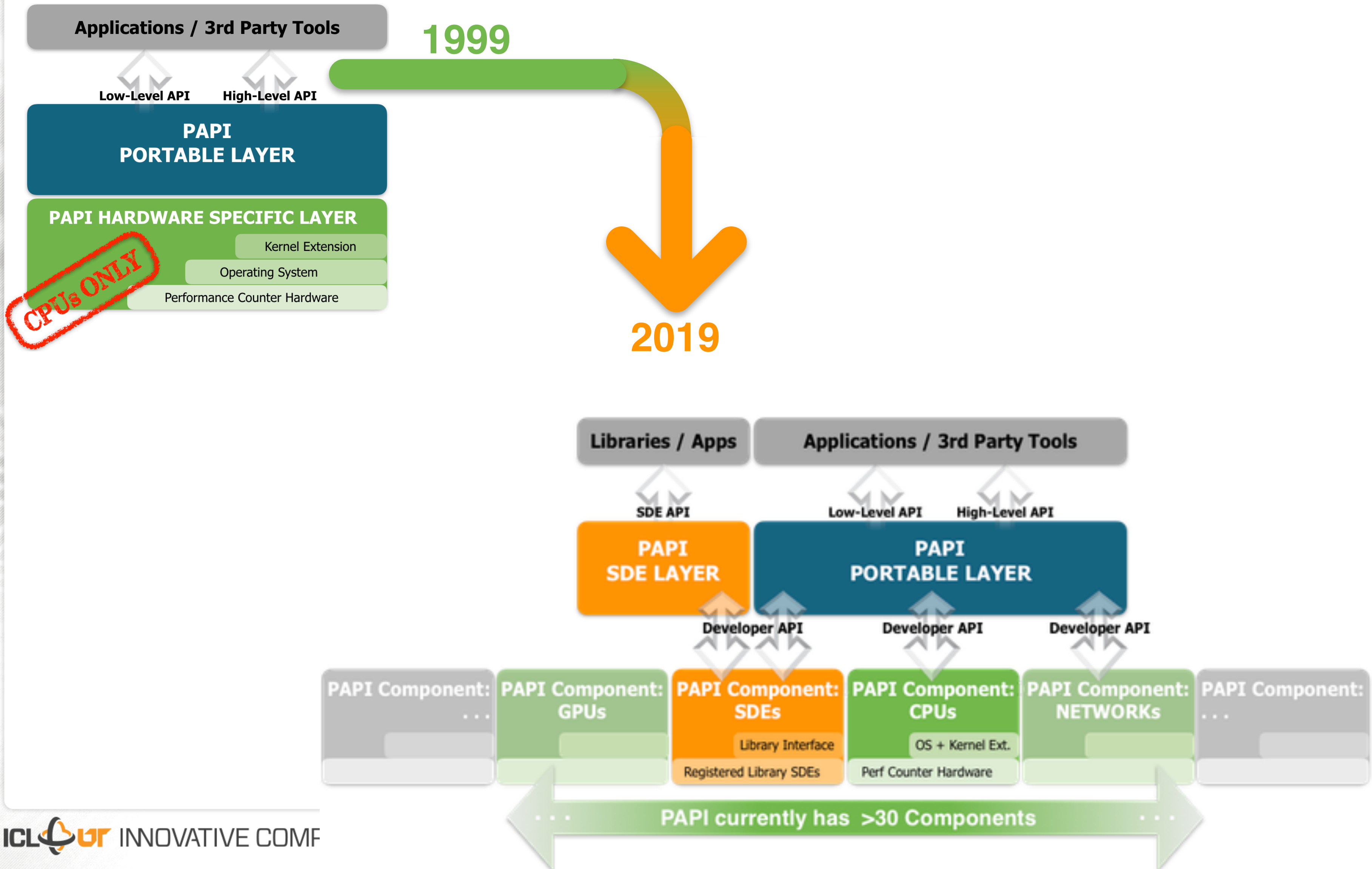
BICG
BICGSTAB
CG
CGS
GMRES
IDR
QMR
TFQMR

Runtime of Solvers:

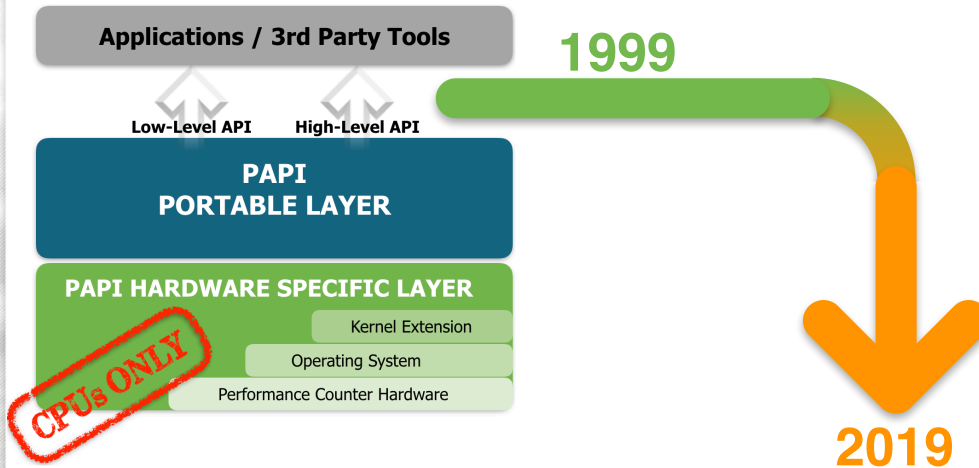
0.1312 sec
0.0956 sec
0.0404 sec
0.1120 sec
0.6497 sec
0.2999 sec
0.1725 sec
0.1910 sec

Matrix from: https://sparse.tamu.edu/HB/662_bus

PAPI Framework: Challenges

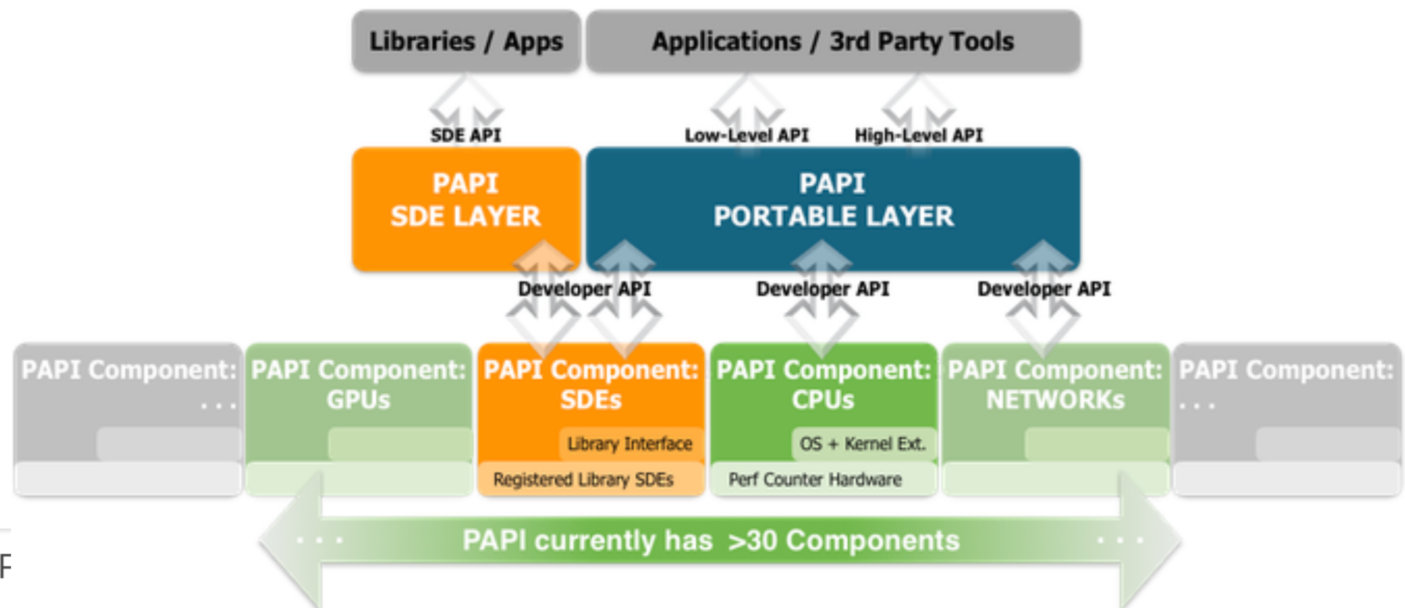


PAPI Framework: Challenges

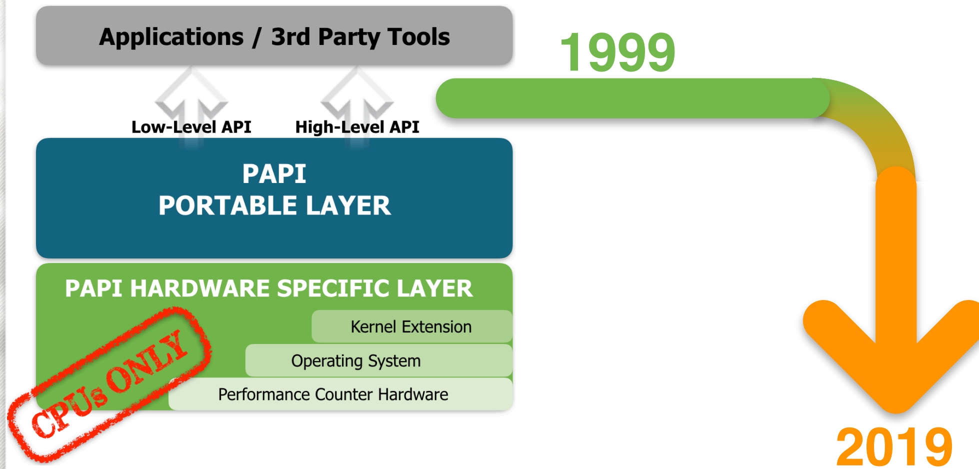


2019

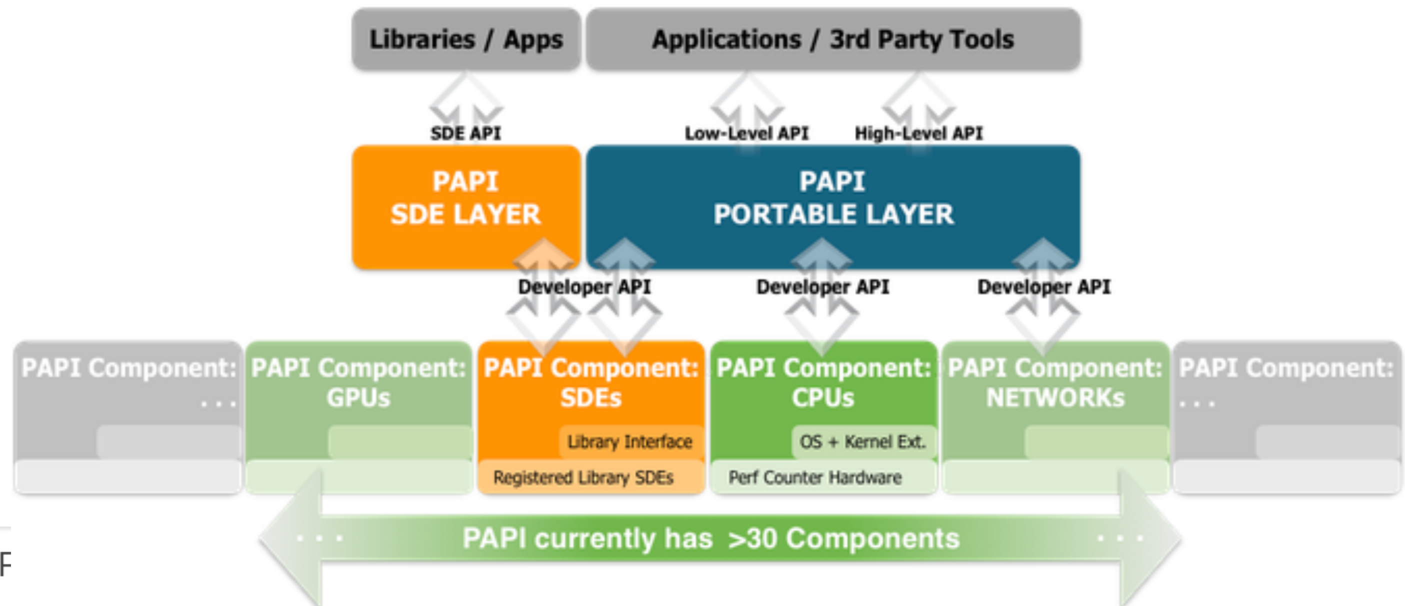
- PAPI has been in constant flux
  - "**extended**" with new CPUs
  - "**redesigned**" for HW Components (GPUs, networks, I/O, power, etc.)
  - "**upgraded**" with SDEs



PAPI Framework: Challenges



- PAPI has been in constant flux
  - "**extended**" with new CPUs
  - "**redesigned**" for HW Components (GPUs, networks, I/O, power, etc.)
  - "**upgraded**" with SDEs
- PAPI's initial intended purpose (CPUs) differs from today's (system + SDEs)
- Counters are not of fixed type anymore
- SDEs go beyond notion of a "counter"



Path forward: PAPI++

Need for a more efficient and flexible software design!

Development of a new C++ performance API (PAPI++) from the ground up:

- Use of modern programming language for **more generic AND safe/easy-to-maintain** code
- PAPI++ code base will be more compact
    - Multiple components will be handled by C++ templating
- Minimize volume of code exposed to changing requirements
    - PAPI's vast test suite will be handled by C++ templating
- Allow for more flexible counter types
- PAPI++ is meant to be PAPI's replacement

3rd Party Tools applying PAPI

ECP Projects

Other Tools not directly part of ECP

ECP DTE (PaRSEC) UTK **http://icl.utk.edu/parsec/**	**ECP LLNL-ATDM (Caliper)** LLVM github.com/LLNL/caliper-compiler	**ECP SNL-ATDM (Kokkos)** SNL **https://github.com/kokkos**	**ECP Proteas (TAU)** University of Oregon **http://tau.uoregon.edu/**
ECP HPCToolkit (HPCToolkit) Rice University **http://hpctoolkit.org**	**Score-P** **http://score-p.org**	**Vampir** TU Dresden **http://www.vampir.eu/**	**Scalasca** FZ Juelich, TU Darmstadt **http://scalasca.org/**
CrayPAT Cray **https://pubs.cray.com/**	**Open|Speedshop** Open|SpeedShop **https://openspeedshop.org/**	**SvPablo** RENCI at UNC **www.renci.org/research/pablo**	**ompP** LMU Munich **http://www.ompp-tool.com/**

INNOVATIVE
COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE UT