You Too Can Do Performance Profiling

Dr Christopher Jones, Dr Marc Paterno

LArSoft Usability Workshop
23 June 2016

Introduction

- Types of Profiling
  - Timing
    - Used to find in which code the program spends most of its time
  - Memory
    - Used to find where memory is being *hoarded*
      - *hoarded*: memory being held for long periods of time
    - Used to find where it is *leaked*
      - *leaked*: memory being ‘`new`’ed but never ‘`delete`’d
    - Used to find where it is being *abused*
      - *abused*: code overwriting a value in memory accidentally
- Tools
  - igprof
    - can do timing and memory (hoarding and leaking) profiling
  - valgrind
    - can do memory hoarding, leaking and abusing profiling
  - Instruments
    - macOS only
    - can do timing and memory (hoarding and leaking) profiling

Steps to Using igprof for Memory Profiling

- Setup the igprof UPS product in your working area

```
setup igprof v5_9_16 -q e9
```

- Run igprof on your lar job

```
igprof -t lar -o igprof_lar.gz lar -c simple.fcl
```

- Process the data gathered by igprof into an sqlite database

```
igprof-analyse --sqlite --demangle -v igprof_lar_optimized.gz  |
sqlite3 igprof_lar_optimized.sql3
```

- Start a web server to easily look at the igprof report

```
igprof-navigator -p 8090 igprof_lar_optimized.sql3 &
```

- Point your web browser to the URL printed out when starting the web server

```
firefox http://test.fnal.gov:8090
```

- Browse the report

Setup the UPS Product

- Setting up igprof makes the command-line igprof executable (and ancillary tools) available.
- igprof is implemented in C++ , so we need to set up the binary compatible version (here, ‘e9’, the same as the LArSoft build you’re using).
- Standard options to use

```
setup igprof v5_9_16 -q e9
```

  `igprof`: setup the UPS product igprof

  `v5_9_16`: is the most recent version of igprof.

  `-q e9`: chooses the e9 qualified build (GCC 4.9.3, using C++14)

- To list the build of igprof installed and available for setup:

```
ups list -aK+ igprof
```

- See http://scisoft.fnal.gov/scisoft/packages/igprof/ for all installable versions

Running igprof

- igprof will run lar for you and monitor the program as it runs

- Documentation
  - Slightly out-dated ones available at main website: http://igprof.org
  - **`igprof -h`** gives information on the command line

- Standard options to use

```
igprof -t lar -o igprof_lar.gz lar -c simple.fcl
```

`-t lar` : only profile programs named ‘lar’

`-o igprof_lar.gz` : write compressed output to file igprof_lar.gz

`lar -c simple.fcl` : run lar normally

Process igprof Results

- The output of igprof is in a form quick to write but not human understandable

- igprof-analyse is used to transform the igprof output to a human usable form
  - a text output is available but takes more effort to understand
  - we will use an sqlite output which can be easily read via a web browser

- Standard options to use

```
igprof-analyse --sqlite --demangle -v igprof_lar.gz  | sqlite3
igprof_lar.sql3
```

`--sqlite` : generate database commands for sqlite

`--demangle` : use human-readable names for C++ functions and classes

`-v` : give feedback as the analysis is running

`igprof_lar.gz` : name of the igprof output file to read

`| sqlite igprof_lar.sql3` : send the output to sqlite which writes a file named igprof_lar.sql3

Start Webserver

- igprof-navigator reads the sqlite file and creates easy to browse web-pages
- Standard options to use

```
igprof-navigator -p 8090 igprof_lar.sql3 &
```

`-p 8090` : specify the network port to use for the web-server

  - any number between 8000-9000 tends to be fine
  - the program gives an error if the port is already in use

igprof_lar.sql3 : name of the sqlite file to use

- igprof-navigator prints out the URL to use by your browser

```
[cdj@test build]$ igprof-navigator -p 8090 igprof_lar.sql3  &
igprof-navigator standalone HTTP server started on port 8090

Point your browser to: http://test.fnal.gov:8090
```

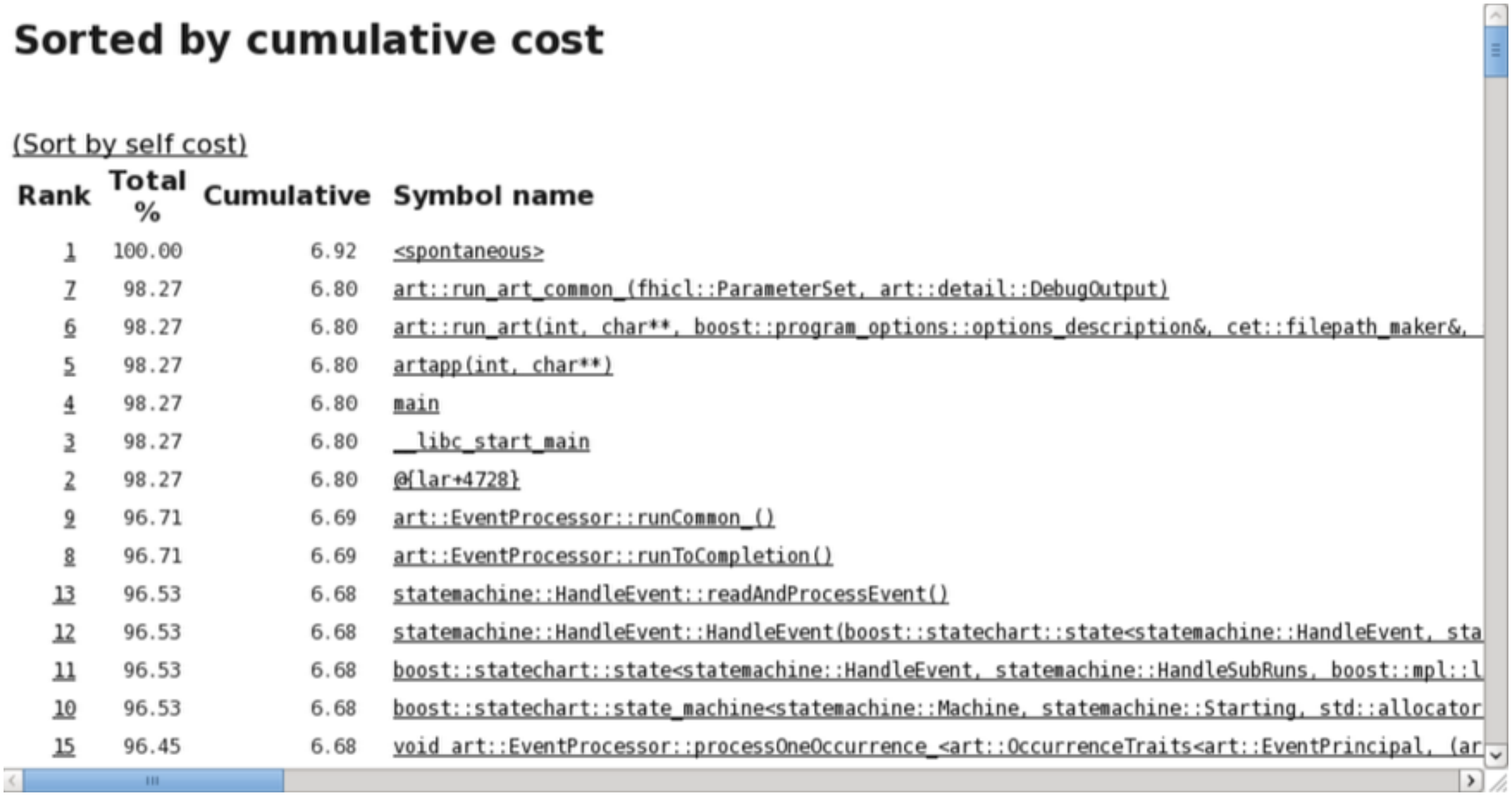
Viewing Webpages

- Web servers started in FNAL network can not be seen outside of the network
  - Often the servers can not be seen outside of the same machine
- Best to run the web browser on same machine as the web server
  - remember to specify the correct port in the URL

```
firefox http://test.fnal.gov:8090
```

Browsing the Web Results

- The URL shows a list of all functions recorded
  - list is ordered by the amount of time the executable spent in the function
  - Cumulative is measured in seconds spend in that function



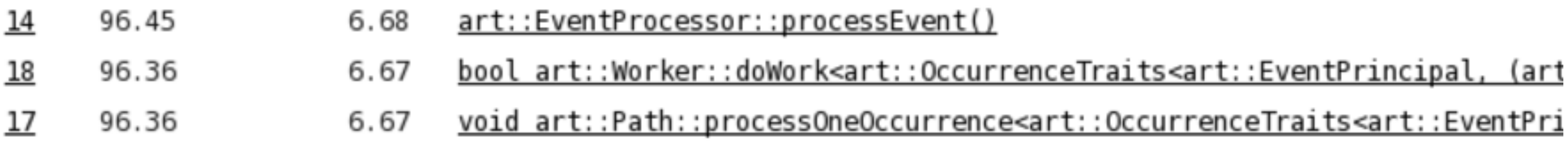
Sorted by cumulative cost

(Sort by self cost)

Rank	Total %	Cumulative	Symbol name
1	100.00	6.92	<spontaneous>
7	98.27	6.80	art::run_art_common_(fhicl::ParameterSet, art::detail::DebugOutput)
6	98.27	6.80	art::run_art(int, char**, boost::program_options::options_description&, cet::filepath_maker&,
5	98.27	6.80	artapp(int, char**)
4	98.27	6.80	main
3	98.27	6.80	__libc_start_main
2	98.27	6.80	0x{lar+4728}
9	96.71	6.69	art::EventProcessor::runCommon_()
8	96.71	6.69	art::EventProcessor::runToCompletion()
13	96.53	6.68	statemachine::HandleEvent::readAndProcessEvent()
12	96.53	6.68	statemachine::HandleEvent::HandleEvent(boost::statechart::state<statemachine::HandleEvent, sta
11	96.53	6.68	boost::statechart::state<statemachine::HandleEvent, statemachine::HandleSubRuns, boost::mpl::l
10	96.53	6.68	boost::statechart::state_machine<statemachine::Machine, statemachine::Starting, std::allocator
15	96.45	6.68	void art::EventProcessor::processOneOccurrence_<art::OccurrenceTraits<art::EventPrincipal, (ar

Where to Start

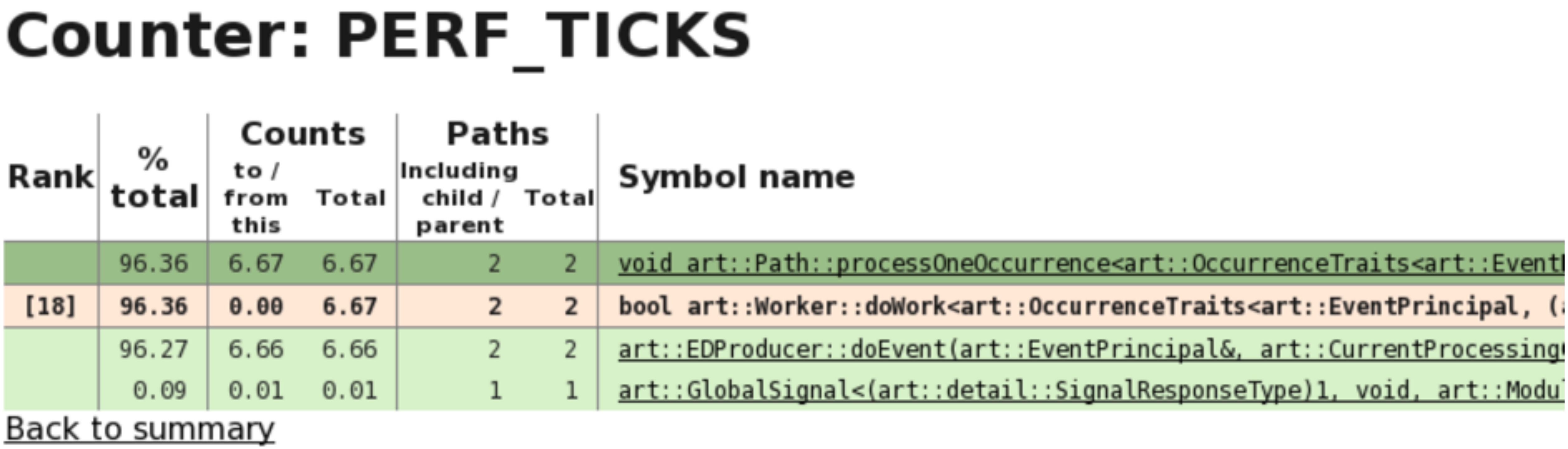
- Many of the top lines are just part of the infrastructure for art
- Want to start at the point where art is calling modules
- Click on the link that begins with bool art::Worker::doWork



```
14    96.45          6.68    art::EventProcessor::processEvent()
18    96.36          6.67    bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (art
17    96.36          6.67    void art::Path::processOneOccurrence<art::OccurrenceTraits<art::EventPri
```

Reading the Report

- This brings up the page for the function art::Worker::doWork



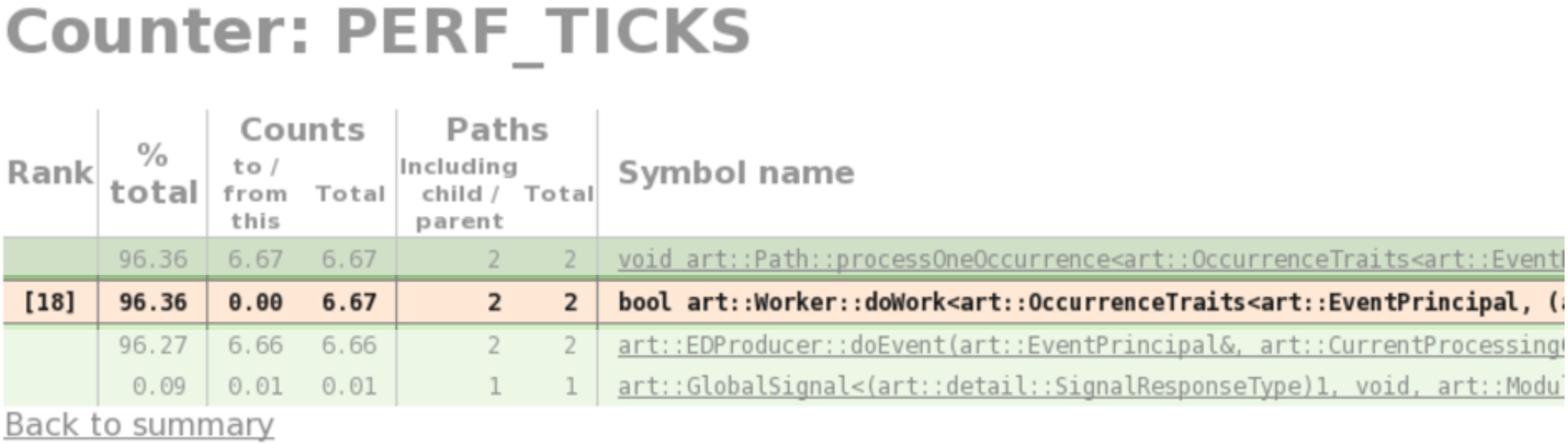
Counter: PERF_TICKS

Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.36	6.67	6.67	2	2	void art::Path::processOneOccurrence<art::OccurrenceTraits<art::Event
[18]	96.36	0.00	6.67	2	2	bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (
	96.27	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing
	0.09	0.01	0.01	1	1	art::GlobalSignal<(art::detail::SignalResponseType)1, void, art::Modu

Back to summary

Reading the Report

- This brings up the page for the function art::Worker::doWork



Counter: PERF_TICKS

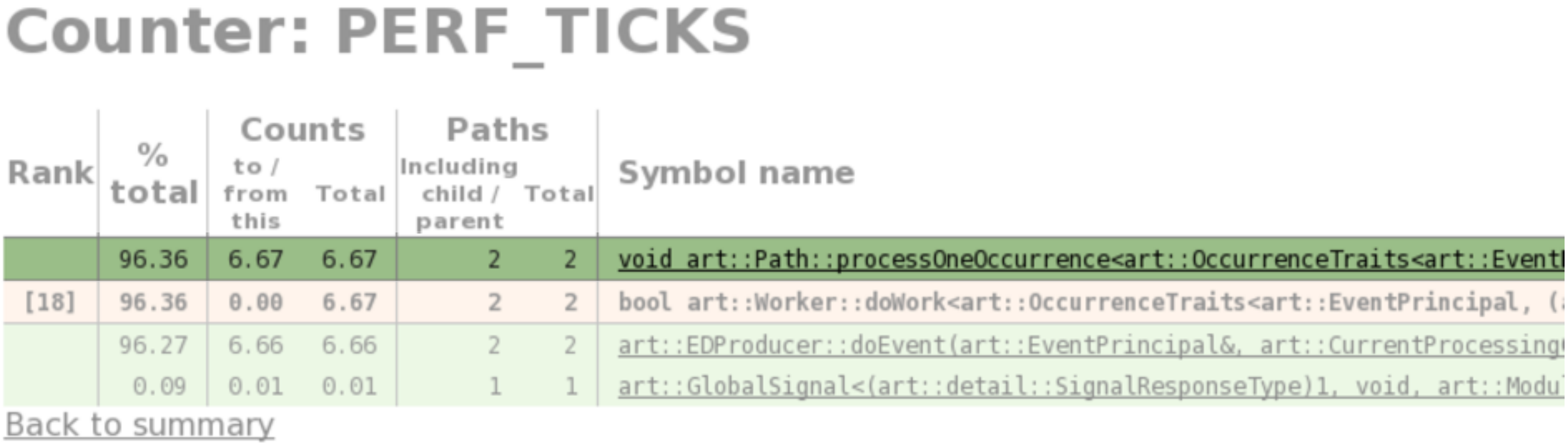
Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.36	6.67	6.67	2	2	void art::Path::processOneOccurrence<art::OccurrenceTraits<art::Event
[18]	**96.36**	**0.00**	**6.67**	**2**	**2**	**bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (**
	96.27	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing
	0.09	0.01	0.01	1	1	art::GlobalSignal<(art::detail::SignalResponseType)1, void, art::Modu

Back to summary

- This row shows the information about the scrutinized function itself

Reading the Report

- This brings up the page for the function art::Worker::doWork



Counter: PERF_TICKS

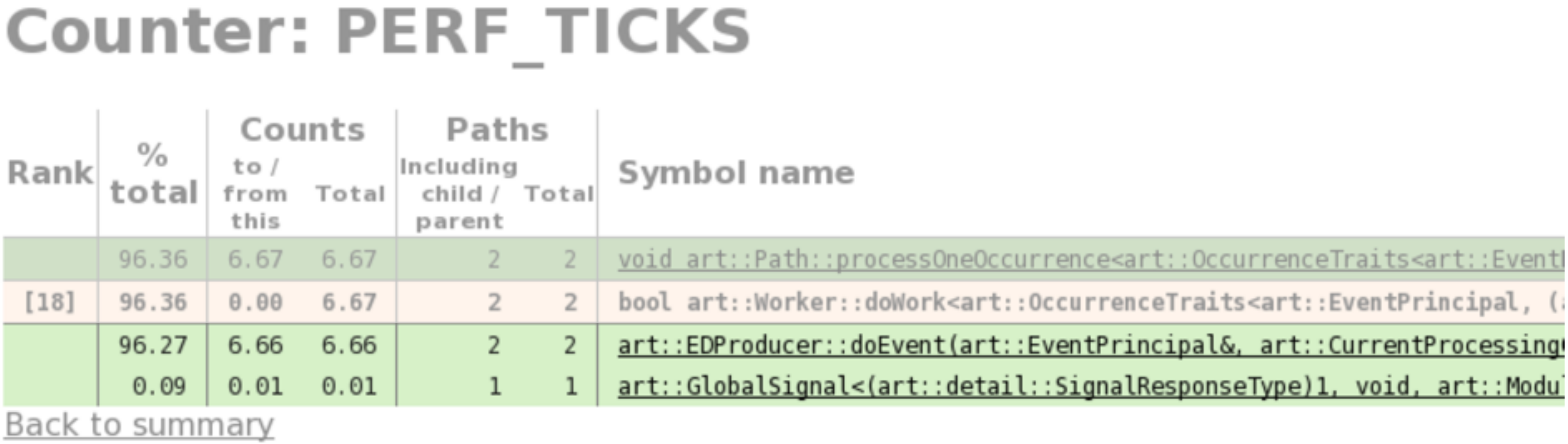
Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.36	6.67	6.67	2	2	void art::Path::processOneOccurrence<art::OccurrenceTraits<art::Event
[18]	96.36	0.00	6.67	2	2	bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (
	96.27	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing
	0.09	0.01	0.01	1	1	art::GlobalSignal<(art::detail::SignalResponseType)1, void, art::Modu

Back to summary

- The row(s) above shows which functions are calling the scrutinized function
  - The top rows are ordered by time

Reading the Report

- This brings up the page for the function art::Worker::doWork



Counter: PERF_TICKS

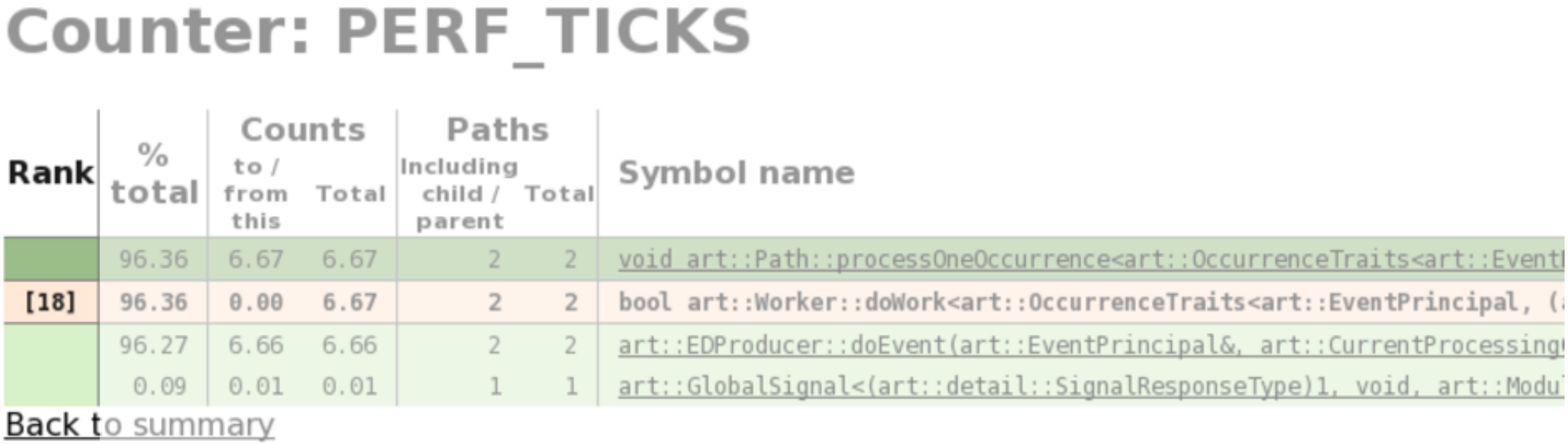
Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.36	6.67	6.67	2	2	void art::Path::processOneOccurrence<art::OccurrenceTraits<art::Event
[18]	96.36	0.00	6.67	2	2	bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (
	96.27	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing
	0.09	0.01	0.01	1	1	art::GlobalSignal<(art::detail::SignalResponseType)1, void, art::Modu

Back to summary

- The rows below show which functions are called by the scrutinized function
    - The bottom rows are ordered by time

Reading the Report

- This brings up the page for the function art::Worker::doWork



Counter: PERF_TICKS

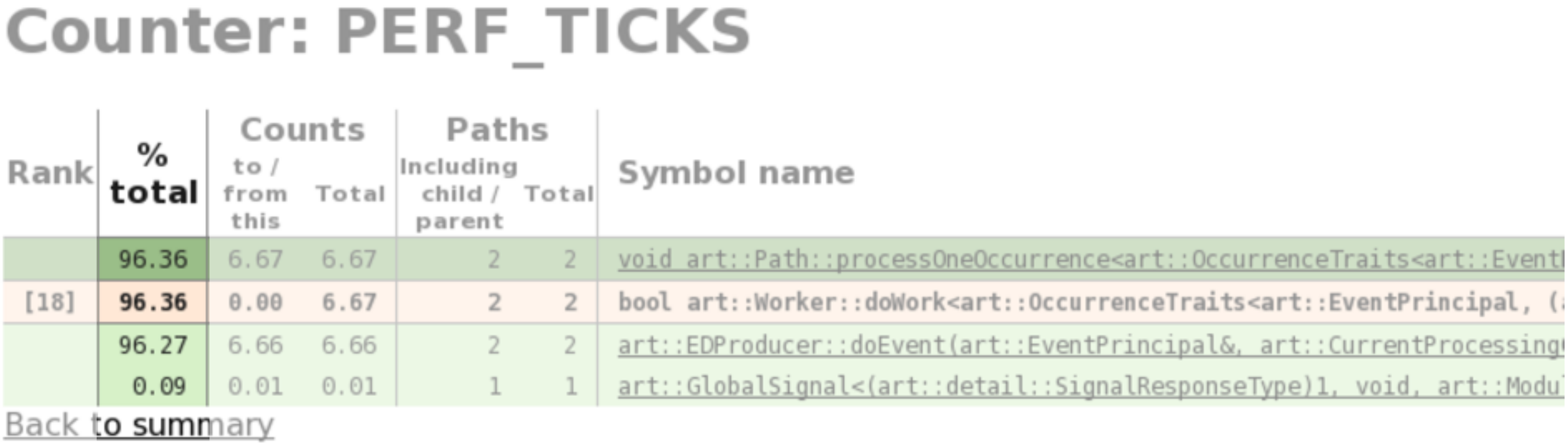
Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.36	6.67	6.67	2	2	void art::Path::processOneOccurrence<art::OccurrenceTraits<art::Event
[18]	96.36	0.00	6.67	2	2	bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (
	96.27	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing
	0.09	0.01	0.01	1	1	art::GlobalSignal<(art::detail::SignalResponseType)1, void, art::Modu

Back to summary

- Column Rank is where in the time ordered list of functions this appears
  - art::Worker::doWork is the 18th most time consuming function in the report

Reading the Report

- This brings up the page for the function art::Worker::doWork



Counter: PERF_TICKS

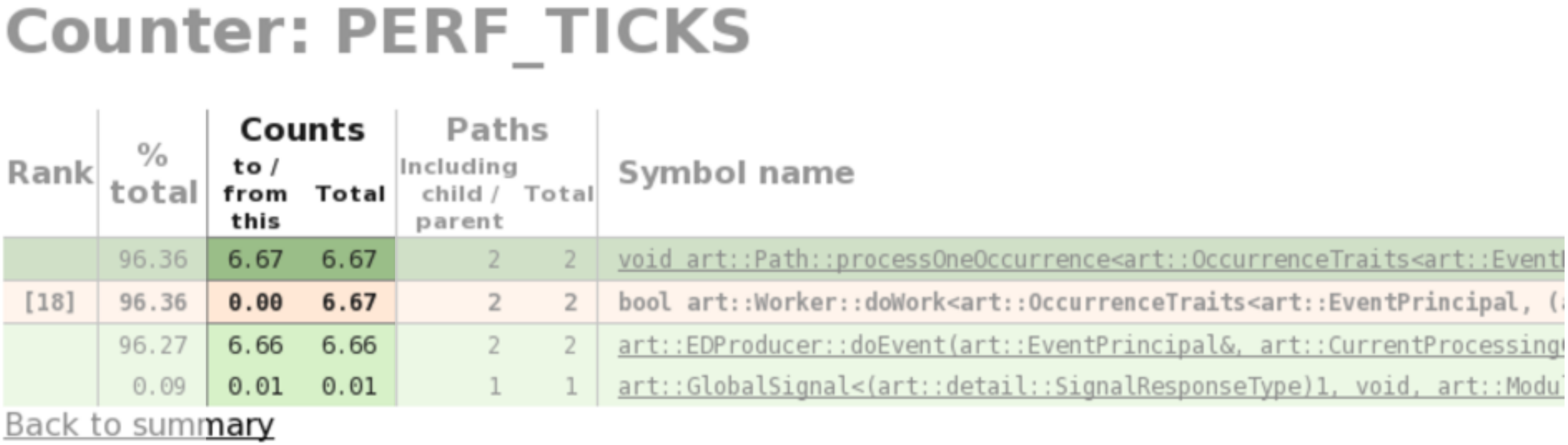
Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.36	6.67	6.67	2	2	void art::Path::processOneOccurrence<art::OccurrenceTraits<art::Event
[18]	96.36	0.00	6.67	2	2	bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (
	96.27	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing
	0.09	0.01	0.01	1	1	art::GlobalSignal<(art::detail::SignalResponseType)1, void, art::Modu

Back to summary

- Column %total is fractional time spent by job in that routine
  - 96.36% of the job time was in art::Worker::doWork

Reading the Report

- This brings up the page for the function art::Worker::doWork



Counter: PERF_TICKS

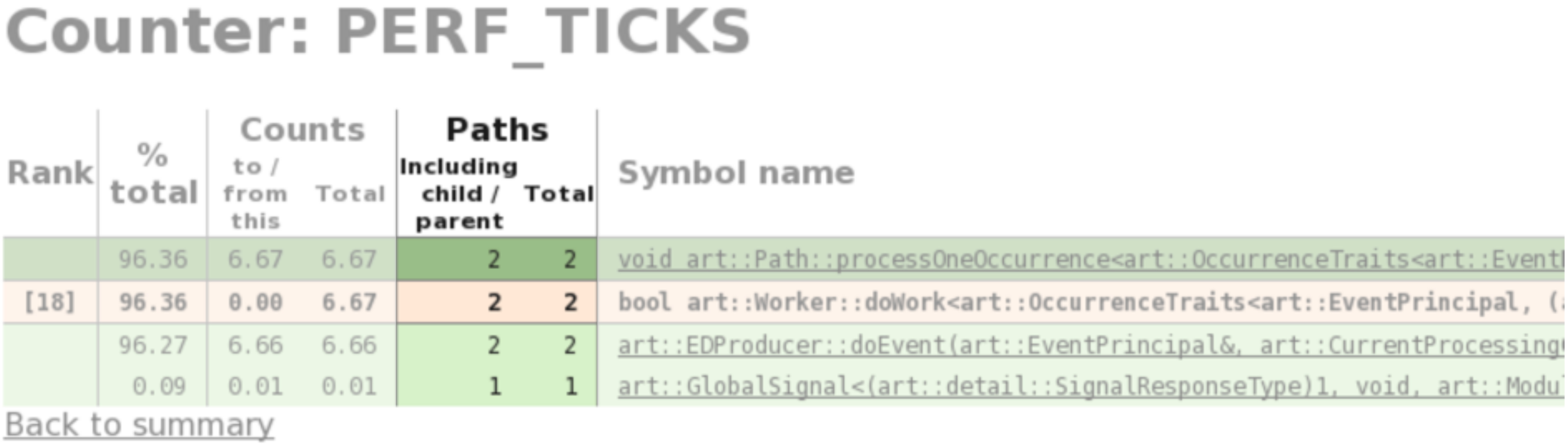
		Counts		Paths		
Rank	% total	to / from this	Total	Including child / parent	Total	Symbol name
	96.36	6.67	6.67	2	2	void art::Path::processOneOccurrence<art::OccurrenceTraits<art::Event
[18]	96.36	0.00	6.67	2	2	bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (
	96.27	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing
	0.09	0.01	0.01	1	1	art::GlobalSignal<(art::detail::SignalResponseType)1, void, art::Modu

Back to summary

- Column Counts says how many seconds spent in the functions
  - to/from for callers is how much time is that function waiting on the scrutinized function
  - to/from for scrutinized function is seconds in that call but not in calling other functions
    - 0.00 seconds is how long art::Worker::doWork is running but not calling other functions
  - to/from for calling functions is time the scrutinized function is waiting for them

Reading the Report

- This brings up the page for the function art::Worker::doWork



Counter: PERF_TICKS

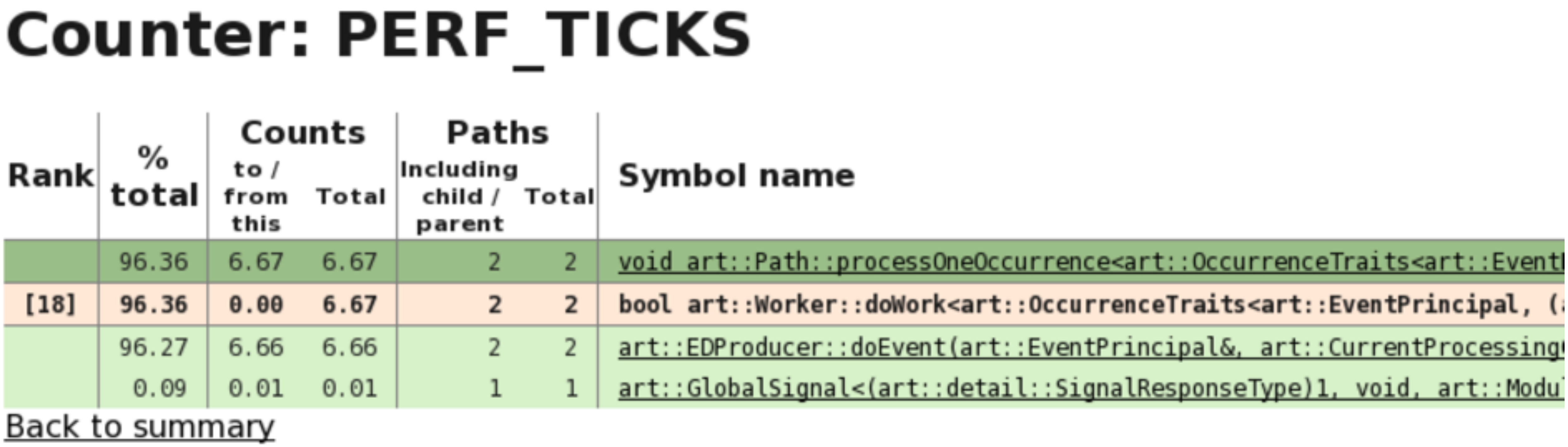
Rank	% total	Counts to / from this	Total	Paths Including child / parent	Total	Symbol name
	96.36	6.67	6.67	2	2	void art::Path::processOneOccurrence<art::OccurrenceTraits<art::Event
[18]	96.36	0.00	6.67	2	2	bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (
	96.27	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing
	0.09	0.01	0.01	1	1	art::GlobalSignal<(art::detail::SignalResponseType)1, void, art::Modu

Back to summary

- Column Paths is not useful for this discussion

Finding Time Consuming Modules

- art::Worker::doWork is how art calls all modules
  - If there were EDAnalyzers, EDFilters or OutputModules in the job they would be shown
- Click on the top most called function art::EDProducer::doEvent



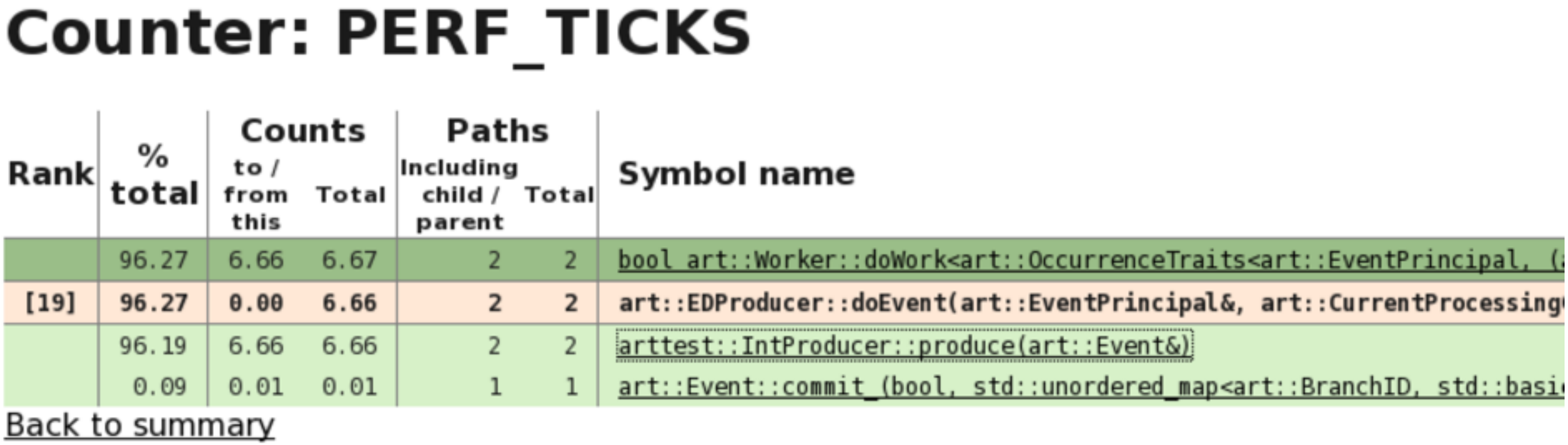
Counter: PERF_TICKS

Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.36	6.67	6.67	2	2	void art::Path::processOneOccurrence<art::OccurrenceTraits<art::EventI
[18]	96.36	0.00	6.67	2	2	bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (
	96.27	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing
	0.09	0.01	0.01	1	1	art::GlobalSignal<(art::detail::SignalResponseType)1, void, art::Modu

Back to summary

Finding Time Consuming Modules (2)

- art::EDProducer::doEvent calls the produce method of all modules
  - All EDProducers called are shown as being called from the function
- arttest::IntProducer is the module we want to analyze
  - Click on its link

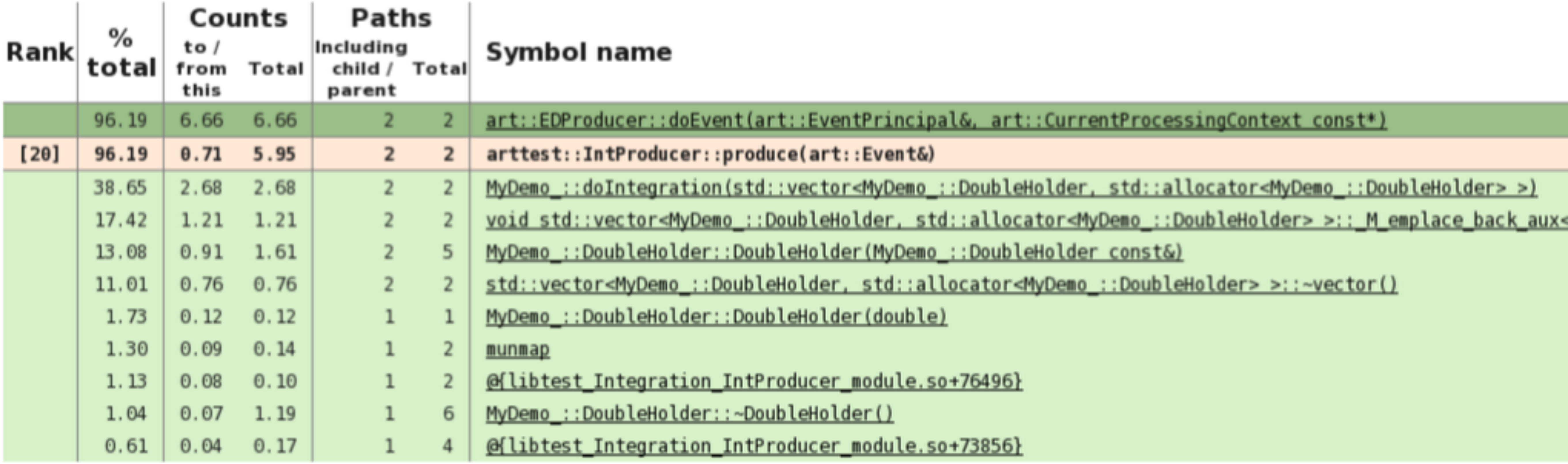


Counter: PERF_TICKS

Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.27	6.66	6.67	2	2	bool art::Worker::doWork<art::OccurrenceTraits<art::EventPrincipal, (
[19]	**96.27**	**0.00**	**6.66**	**2**	**2**	**art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessing**
	96.19	6.66	6.66	2	2	arttest::IntProducer::produce(art::Event&)
	0.09	0.01	0.01	1	1	art::Event::commit_(bool, std::unordered_map<art::BranchID, std::basi

Back to summary

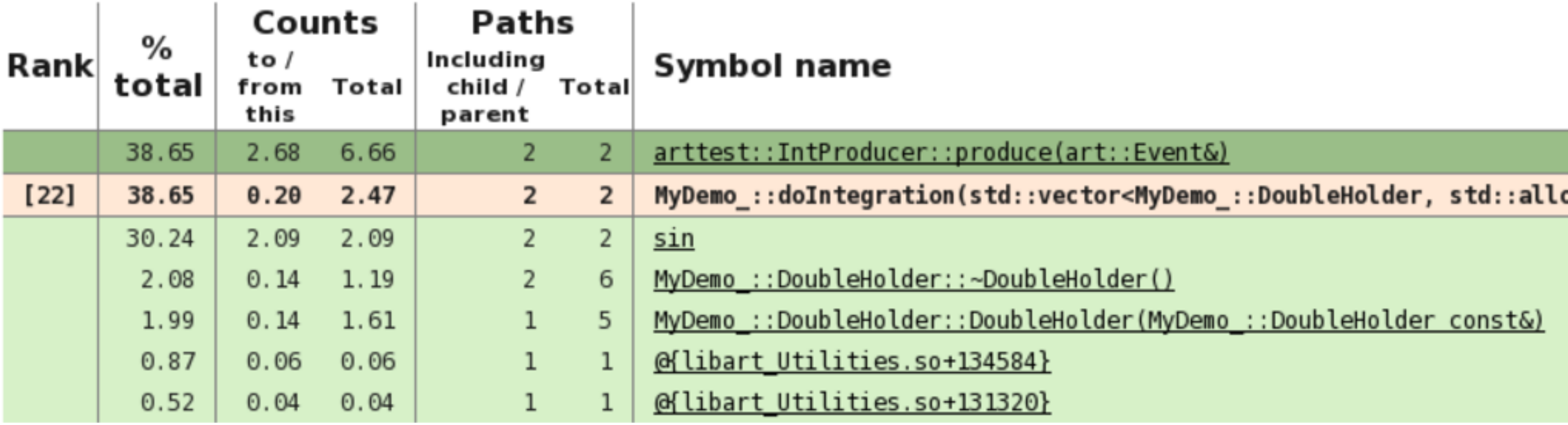
IntProducer Report



Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.19	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessingContext const*)
[20]	96.19	0.71	5.95	2	2	arttest::IntProducer::produce(art::Event&)
	38.65	2.68	2.68	2	2	MyDemo_::doIntegration(std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >)
	17.42	1.21	1.21	2	2	void std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::_M_emplace_back_aux<
	13.08	0.91	1.61	2	5	MyDemo_::DoubleHolder::DoubleHolder(MyDemo_::DoubleHolder const&)
	11.01	0.76	0.76	2	2	std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::~vector()
	1.73	0.12	0.12	1	1	MyDemo_::DoubleHolder::DoubleHolder(double)
	1.30	0.09	0.14	1	2	munmap
	1.13	0.08	0.10	1	2	@{libtest_Integration_IntProducer_module.so+76496}
	1.04	0.07	1.19	1	6	MyDemo_::DoubleHolder::~DoubleHolder()
	0.61	0.04	0.17	1	4	@{libtest_Integration_IntProducer_module.so+73856}

- The longest call is to MyDemo_::doIntegration
    - that is where the work of the module gets done
- Nearly half the time is in dealing with MyDemo_::DoubleHolder!
    - calls to std::vector<MyDemo_::DoubleHolder>
    - constructing and copying MyDemo_::DoubleHolders

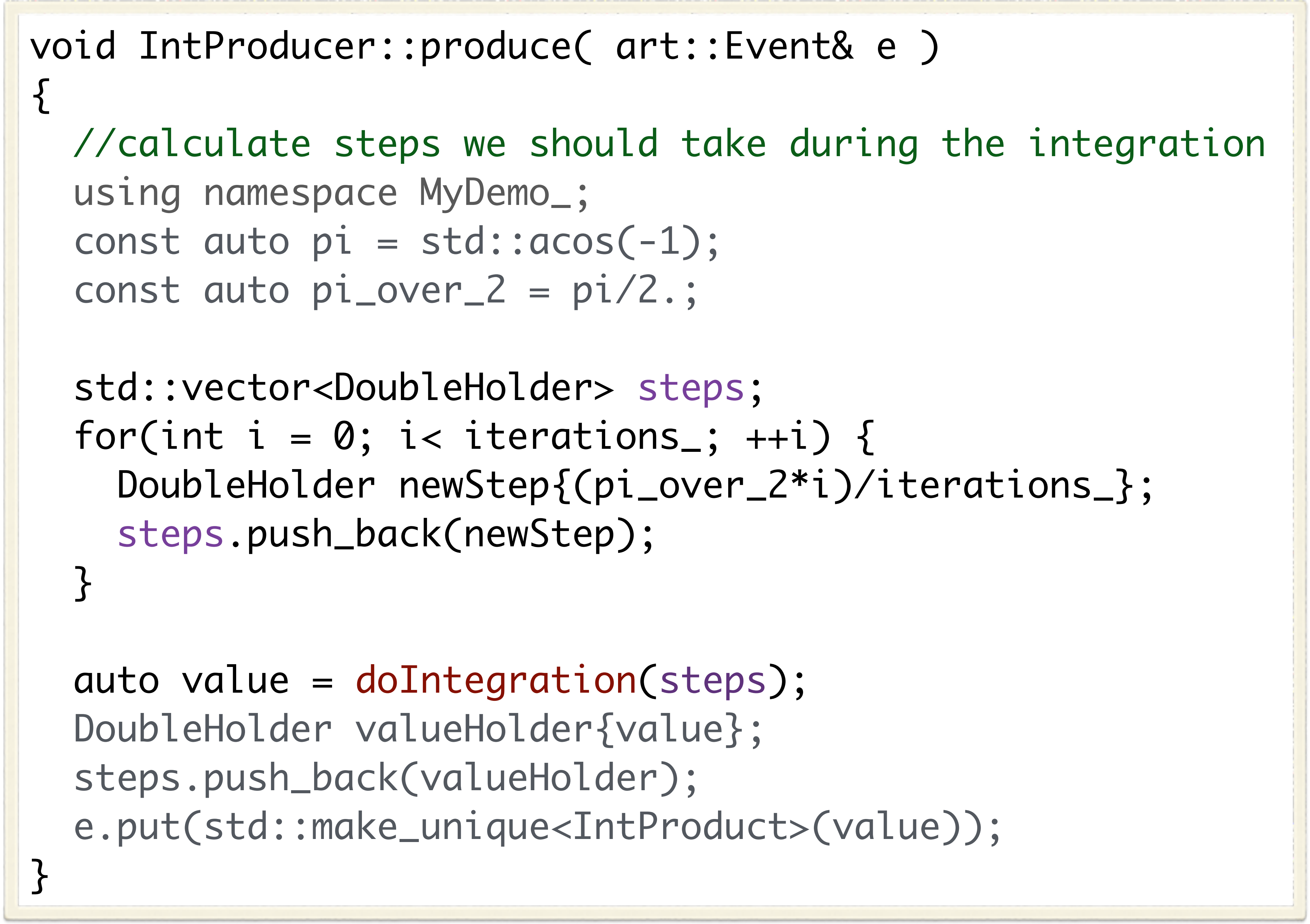
IntProducer Report (2)



Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	38.65	2.68	6.66	2	2	arttest::IntProducer::produce(art::Event&)
[22]	38.65	0.20	2.47	2	2	MyDemo_::doIntegration(std::vector<MyDemo_::DoubleHolder, std::allo
	30.24	2.09	2.09	2	2	sin
	2.08	0.14	1.19	2	6	MyDemo_::DoubleHolder::~DoubleHolder()
	1.99	0.14	1.61	1	5	MyDemo_::DoubleHolder::DoubleHolder(MyDemo_::DoubleHolder const&)
	0.87	0.06	0.06	1	1	@{libart_Utilities.so+134584}
	0.52	0.04	0.04	1	1	@{libart_Utilities.so+131320}

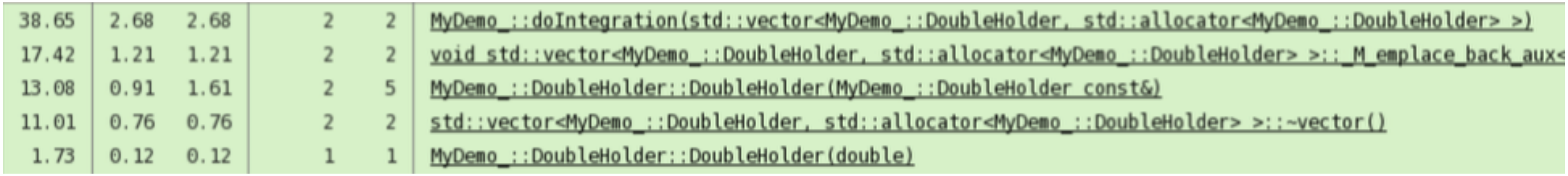
- Most of the time in sin
- A smaller fraction is constructing and destructing MyDemo_::DoubleHolder

Code of IntProducer

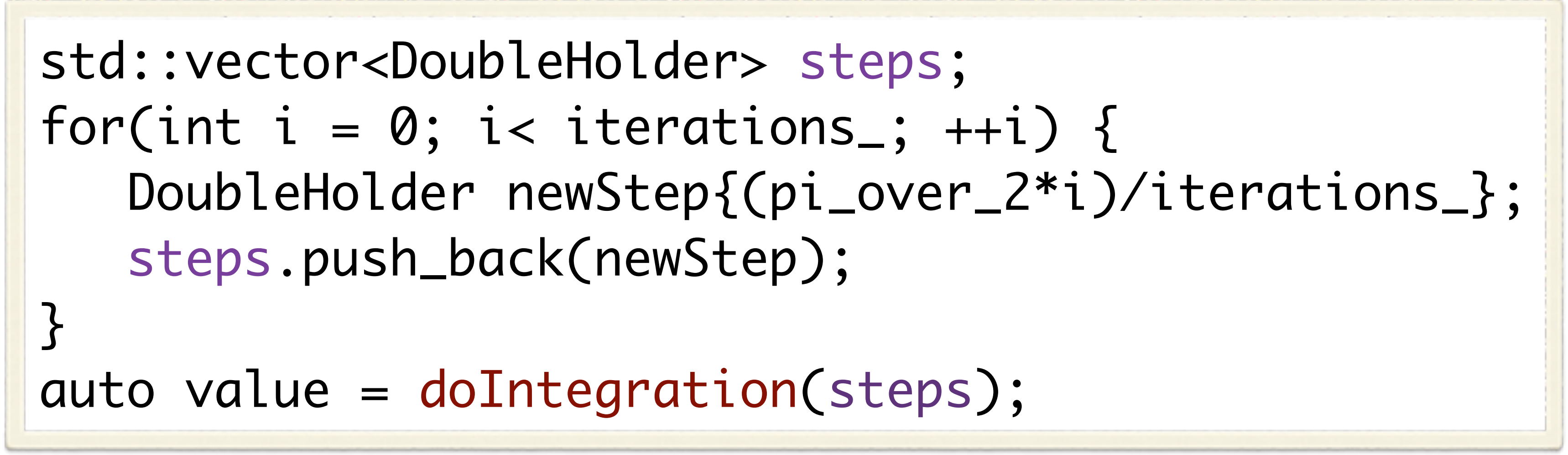


```
void IntProducer::produce( art::Event& e )
{
  //calculate steps we should take during the integration
  using namespace MyDemo_;
  const auto pi = std::acos(-1);
  const auto pi_over_2 = pi/2.;

  std::vector<DoubleHolder> steps;
  for(int i = 0; i< iterations_; ++i) {
    DoubleHolder newStep{(pi_over_2*i)/iterations_};
    steps.push_back(newStep);
  }

  auto value = doIntegration(steps);
  DoubleHolder valueHolder{value};
  steps.push_back(valueHolder);
  e.put(std::make_unique<IntProduct>(value));
}
```

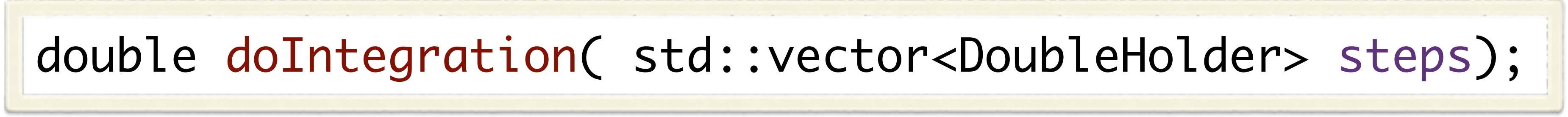
Time Spent In std::vector



```
38.65 | 2.68   2.68 |      2      2 | MyDemo_::doIntegration(std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >)
17.42 | 1.21   1.21 |      2      2 | void std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::_M_emplace_back_aux<
13.08 | 0.91   1.61 |      2      5 | MyDemo_::DoubleHolder::DoubleHolder(MyDemo_::DoubleHolder const&)
11.01 | 0.76   0.76 |      2      2 | std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::~vector()
 1.73 | 0.12   0.12 |      1      1 | MyDemo_::DoubleHolder::DoubleHolder(double)
```

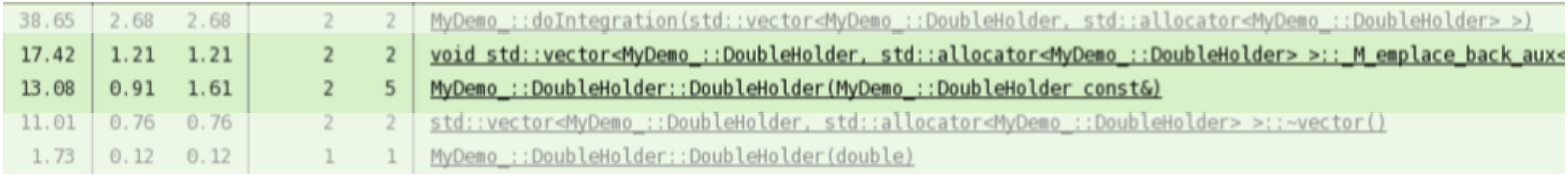


```
std::vector<DoubleHolder> steps;
for(int i = 0; i< iterations_; ++i) {
   DoubleHolder newStep{(pi_over_2*i)/iterations_};
   steps.push_back(newStep);
}
auto value = doIntegration(steps);
```

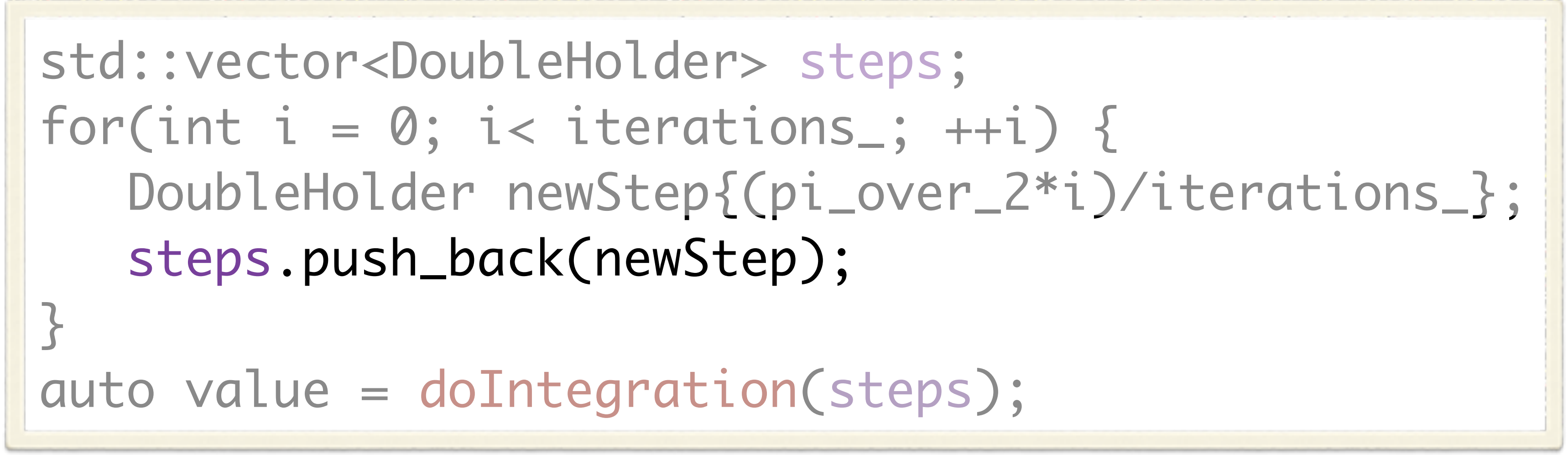


```
double doIntegration( std::vector<DoubleHolder> steps);
```

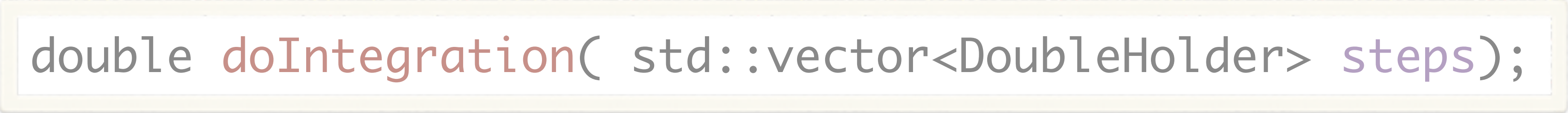
Time Spent In std::vector



38.65	2.68	2.68	2	2	MyDemo_::doIntegration(std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >)
17.42	1.21	1.21	2	2	void std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::_M_emplace_back_aux<
13.08	0.91	1.61	2	5	MyDemo_::DoubleHolder::DoubleHolder(MyDemo_::DoubleHolder const&)
11.01	0.76	0.76	2	2	std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::~vector()
1.73	0.12	0.12	1	1	MyDemo_::DoubleHolder::DoubleHolder(double)



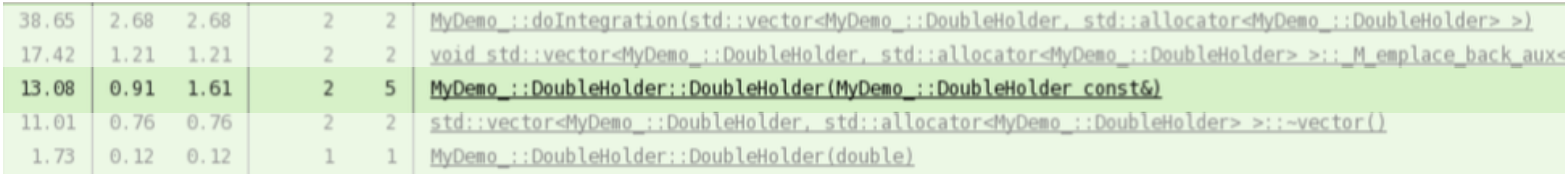
```
std::vector<DoubleHolder> steps;
for(int i = 0; i< iterations_; ++i) {
   DoubleHolder newStep{(pi_over_2*i)/iterations_};
   steps.push_back(newStep);
}
auto value = doIntegration(steps);
```



```
double doIntegration( std::vector<DoubleHolder> steps);
```

- push_back into vector accounts for these
  - copy constructor also called when vector has to grow its memory
- fix: use reserve since know exactly how many items to insert

Time Spent In std::vector



38.65	2.68	2.68	2	2	MyDemo_::doIntegration(std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >)
17.42	1.21	1.21	2	2	void std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::_M_emplace_back_aux<
13.08	**0.91**	**1.61**	**2**	**5**	**MyDemo_::DoubleHolder::DoubleHolder(MyDemo_::DoubleHolder const&)**
11.01	0.76	0.76	2	2	std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::~vector()
1.73	0.12	0.12	1	1	MyDemo_::DoubleHolder::DoubleHolder(double)

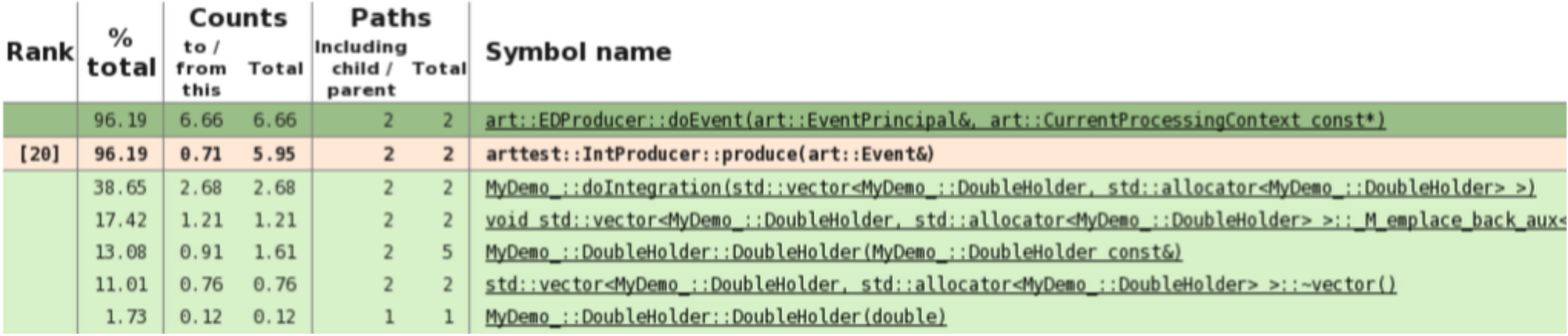
```
std::vector<DoubleHolder> steps;
for(int i = 0; i< iterations_; ++i) {
   DoubleHolder newStep{(pi_over_2*i)/iterations_};
   steps.push_back(newStep);
}
auto value = doIntegration(steps);
```

```
double doIntegration( std::vector<DoubleHolder> steps);
```

- Passing arguments by value account for many copy constructor calls
- Fix: change function to use const reference

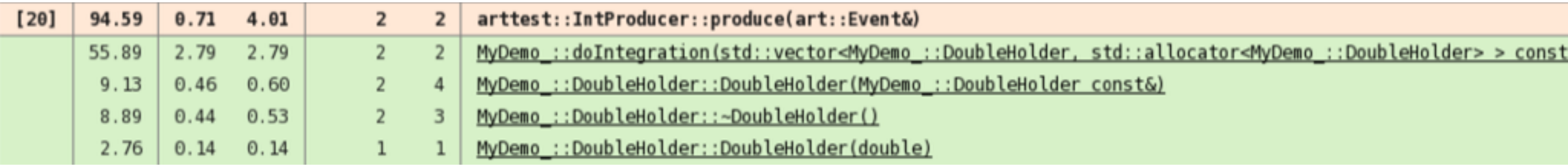
Results After Optimizing for std::vector

- Original Result



Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.19	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessingContext const*)
[20]	**96.19**	**0.71**	**5.95**	**2**	**2**	**arttest::IntProducer::produce(art::Event&)**
	38.65	2.68	2.68	2	2	MyDemo_::doIntegration(std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >)
	17.42	1.21	1.21	2	2	void std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::_M_emplace_back_aux<
	13.08	0.91	1.61	2	5	MyDemo_::DoubleHolder::DoubleHolder(MyDemo_::DoubleHolder const&)
	11.01	0.76	0.76	2	2	std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::~vector()
	1.73	0.12	0.12	1	1	MyDemo_::DoubleHolder::DoubleHolder(double)

- New Result

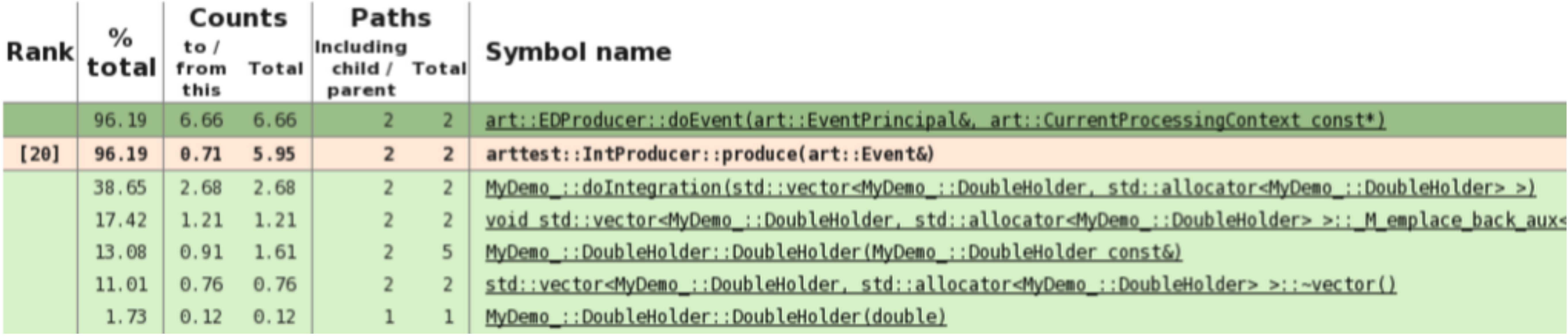


[20]	**94.59**	**0.71**	**4.01**	**2**	**2**	**arttest::IntProducer::produce(art::Event&)**
	55.89	2.79	2.79	2	2	MyDemo_::doIntegration(std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> > const
	9.13	0.46	0.60	2	4	MyDemo_::DoubleHolder::DoubleHolder(MyDemo_::DoubleHolder const&)
	8.89	0.44	0.53	2	3	MyDemo_::DoubleHolder::~DoubleHolder()
	2.76	0.14	0.14	1	1	MyDemo_::DoubleHolder::DoubleHolder(double)

- Speed Improvement: 41%
  - original time: 6.66s
  - new time: 4.72
- Next Step: Inline constructor/destructor for MyDemo_::DoubleHolder

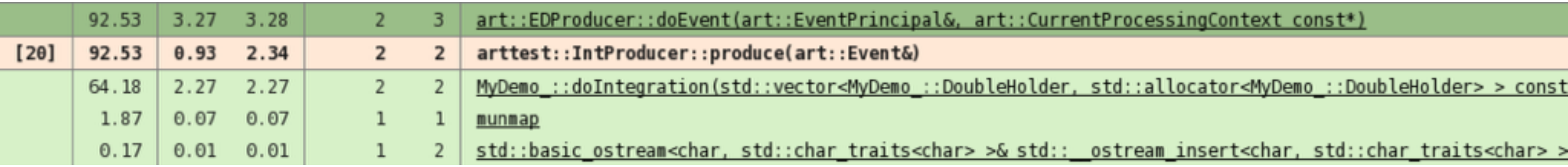
Results After Inlining

- Original Result



Rank	% total	Counts		Paths		Symbol name
		to / from this	Total	Including child / parent	Total	
	96.19	6.66	6.66	2	2	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessingContext const*)
[20]	**96.19**	**0.71**	**5.95**	**2**	**2**	**arttest::IntProducer::produce(art::Event&)**
	38.65	2.68	2.68	2	2	MyDemo_::doIntegration(std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >)
	17.42	1.21	1.21	2	2	void std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::_M_emplace_back_aux<
	13.08	0.91	1.61	2	5	MyDemo_::DoubleHolder::DoubleHolder(MyDemo_::DoubleHolder const&)
	11.01	0.76	0.76	2	2	std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> >::~vector()
	1.73	0.12	0.12	1	1	MyDemo_::DoubleHolder::DoubleHolder(double)

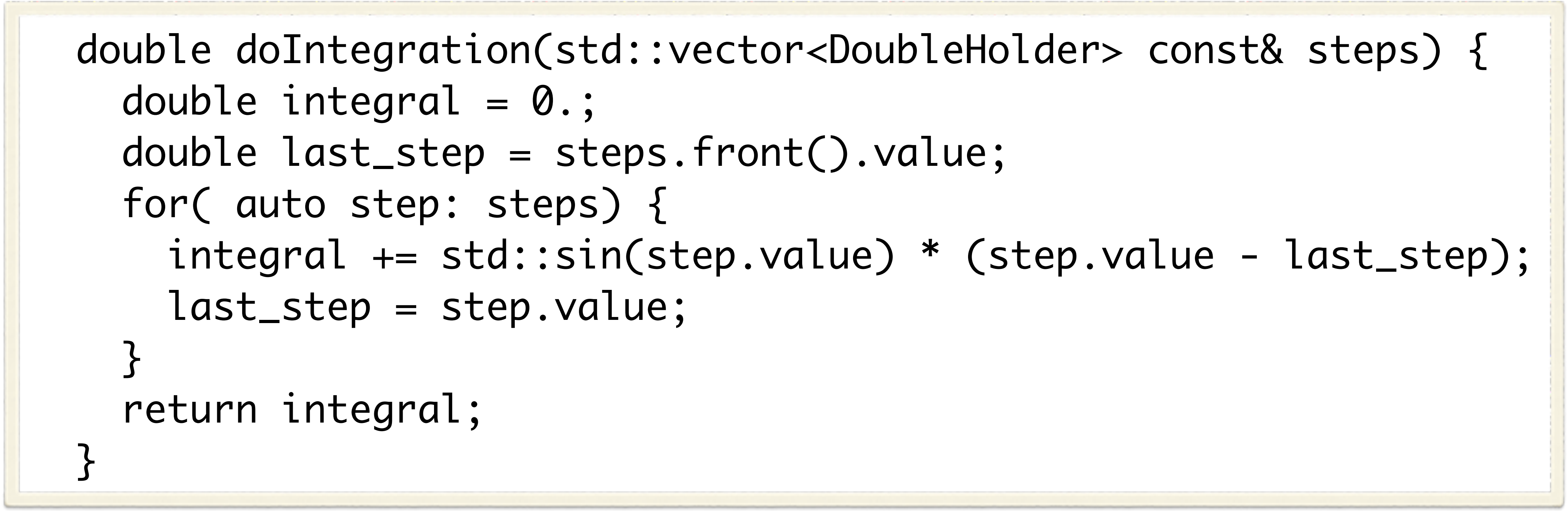
- Final Result



	92.53	3.27	3.28	2	3	art::EDProducer::doEvent(art::EventPrincipal&, art::CurrentProcessingContext const*)
[20]	**92.53**	**0.93**	**2.34**	**2**	**2**	**arttest::IntProducer::produce(art::Event&)**
	64.18	2.27	2.27	2	2	MyDemo_::doIntegration(std::vector<MyDemo_::DoubleHolder, std::allocator<MyDemo_::DoubleHolder> > const
	1.87	0.07	0.07	1	1	munmap
	0.17	0.01	0.01	1	2	std::basic_ostream<char, std::char_traits<char> >& std::__ostream_insert<char, std::char_traits<char> >

- Speed Improvement: 200%
  - original time: 6.66 s
  - final time: 3.27
- Take-home: the more the compiler can see, the more it can optimize

Change the Algorithm



```
double doIntegration(std::vector<DoubleHolder> const& steps) {
  double integral = 0.;
  double last_step = steps.front().value;
  for( auto step: steps) {
    integral += std::sin(step.value) * (step.value - last_step);
    last_step = step.value;
  }
  return integral;
}
```

- doIntegral is doing a numeric integration of sin
  - This is could be done analytically instead
- Take-home: often the best performance increase are from a new algorithm

Conclusion

- igprof has been found to be a useful tool for LArSoft performance analysis
- Translating results from performance to which line of code is tricky
  - The code might have been inlined so is not seen or timing goes to an indirect call
  - Compiler may implicitly add additional calls
    - E.g. passing function arguments by value will invoke constructors
  - Performance reviews are an iterative process
    - measure, change code, repeat
  - Often the greatest timing performance comes from a change of algorithm/data structures