Boosting Evolution:
Scaling Erlang up to 64 cores

Piotr Anielski, Jan Stypka

AGH University of Science and Technology
Department of Computer Science
Kraków, Poland

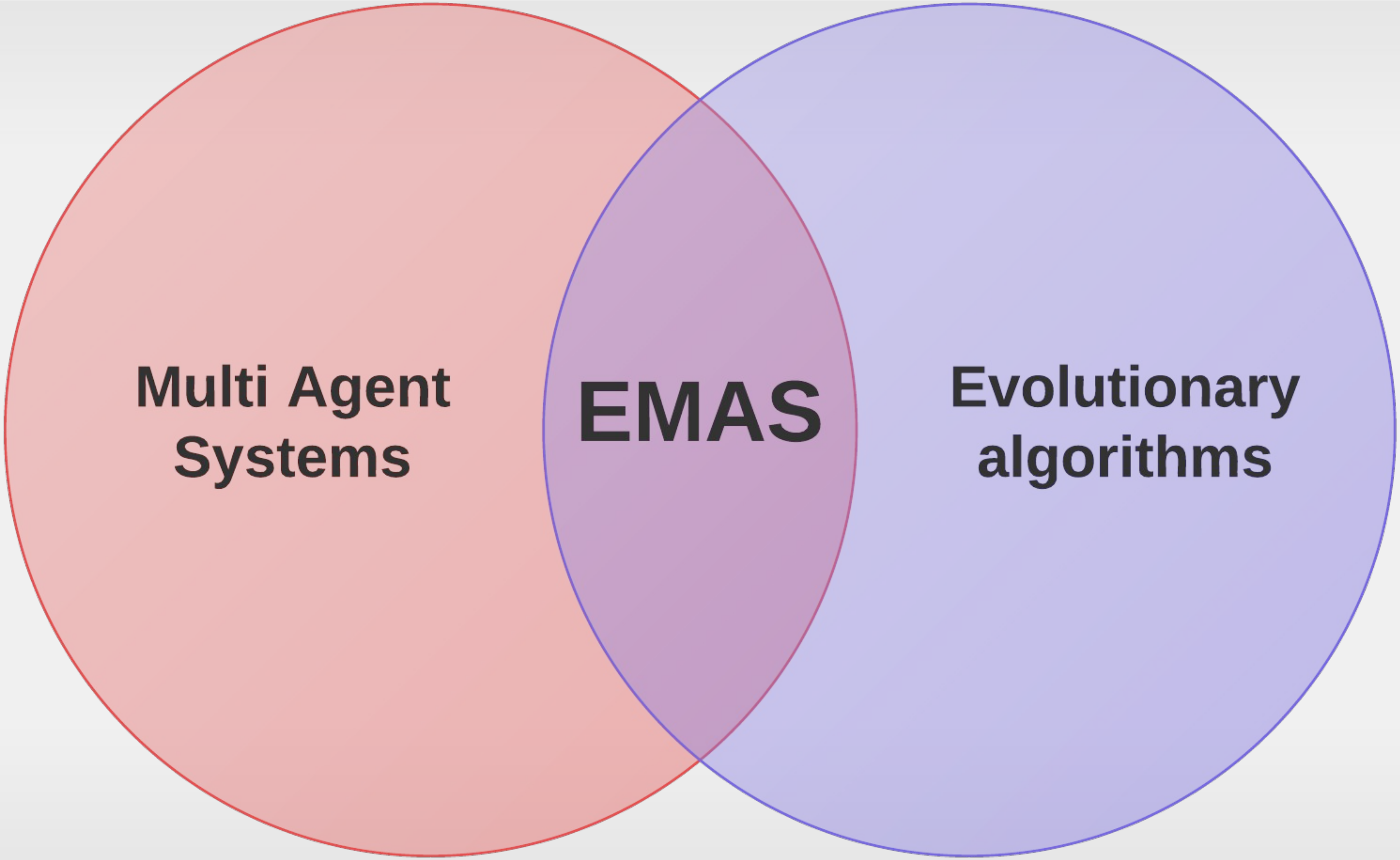
Agenda

1) Algorithm description
2) Different implementations
3) Scaling challenges and applied solutions
4) Summary

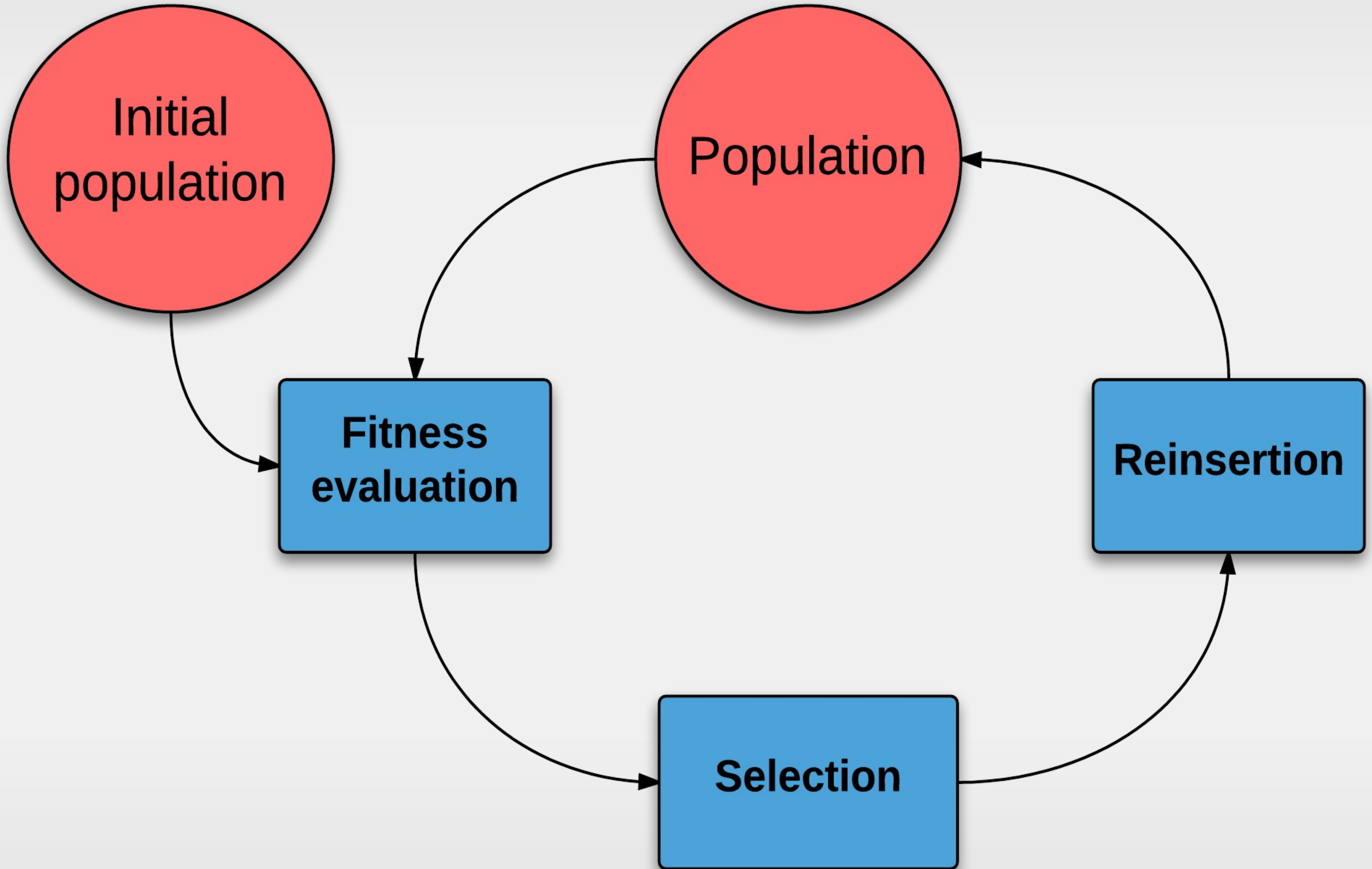
EMAS

Evolutionary Multi Agent System

EMAS



Evolutionary Algorithm

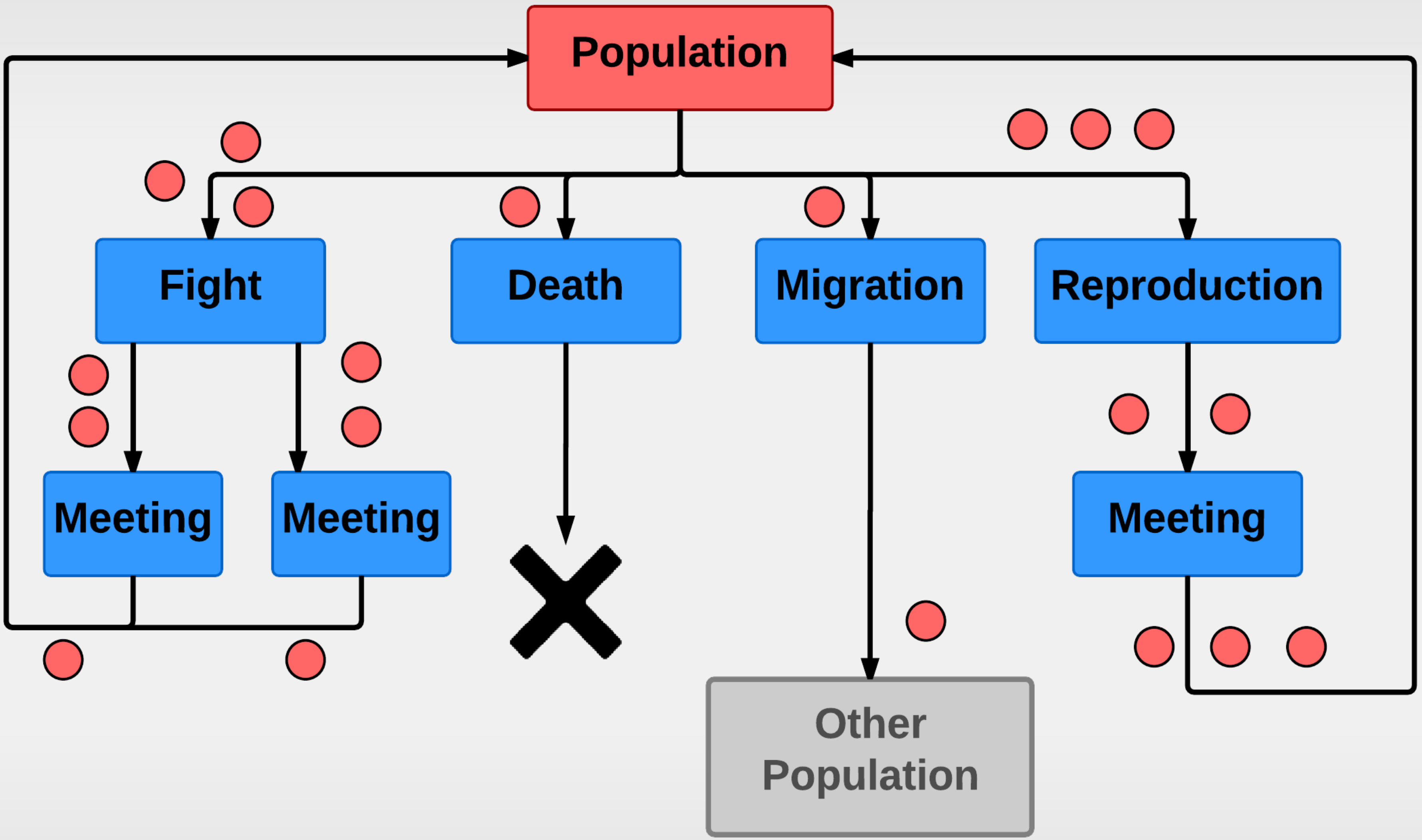


EMAS

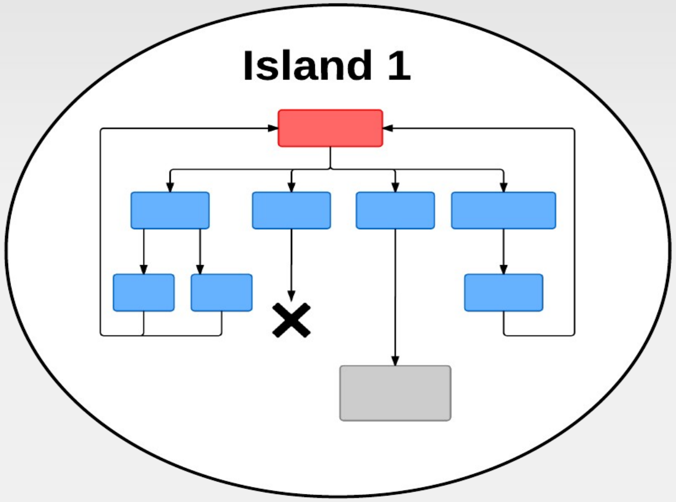
- Optimization algorithm
  - Aims to avoid local optima
  - e.g. for solving scheduling problems
- Autonomous agents
- Decentralized computation
- Easily parallelizable

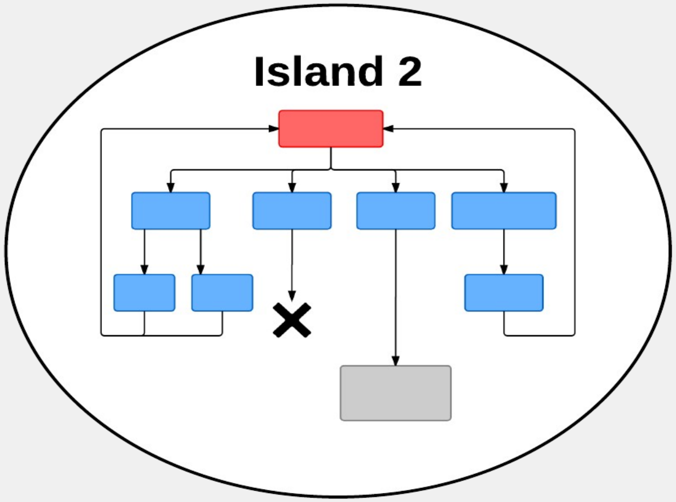
EMAS – how it works

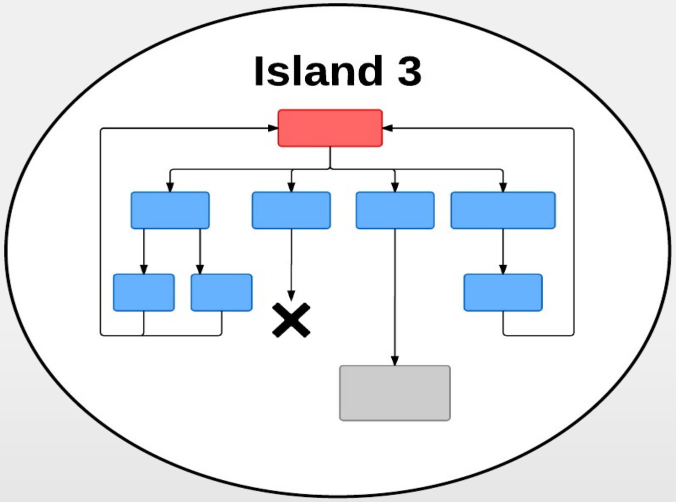
INTELLIGENT INFORMATION SYSTEMS GROUP



EMAS – islands





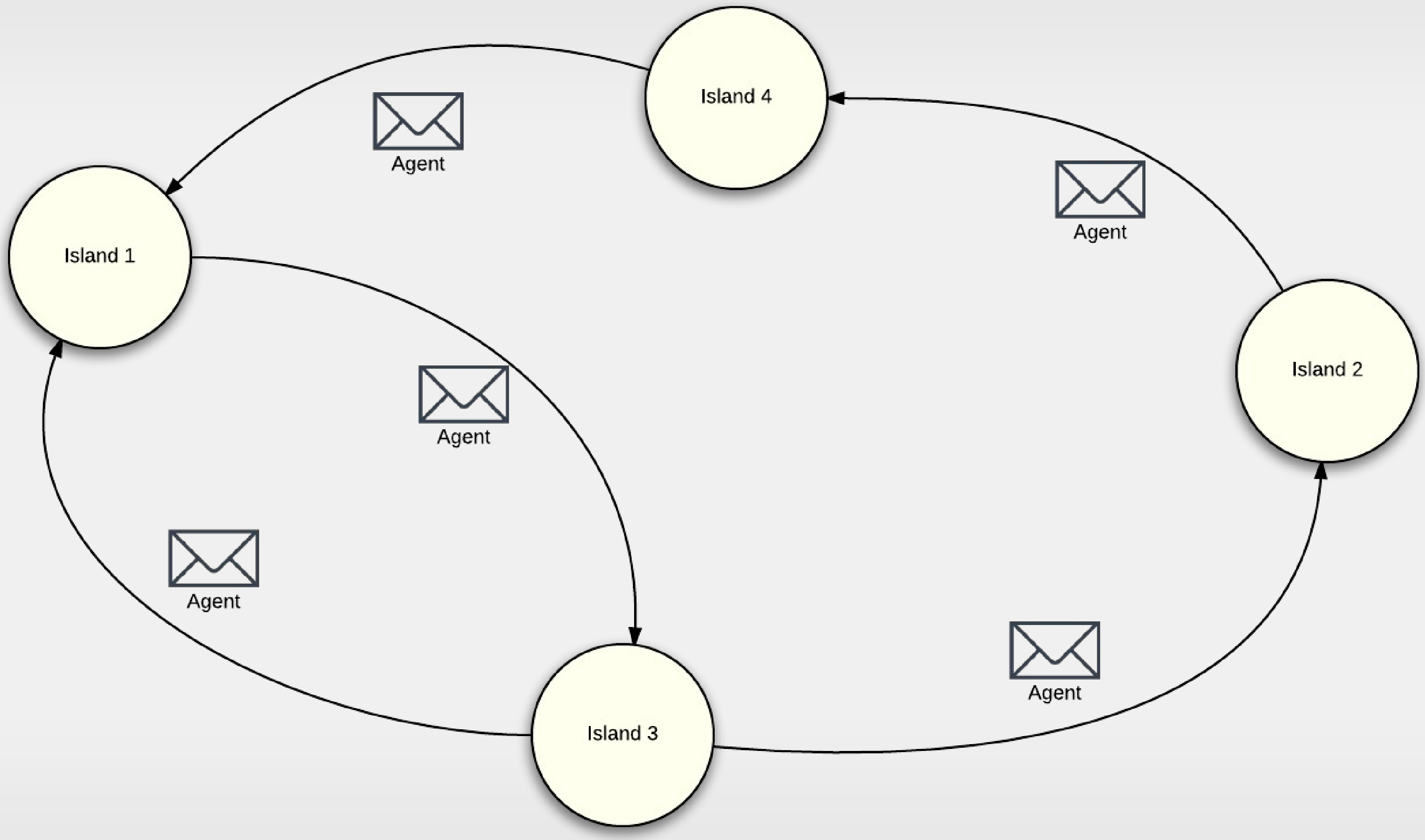


Implementations

- Sequential
- Hybrid (coarse-grained)
- Concurrent (fine-grained)
- Skel (coarse-grained)

Everything implemented in Erlang

Coarse-grained



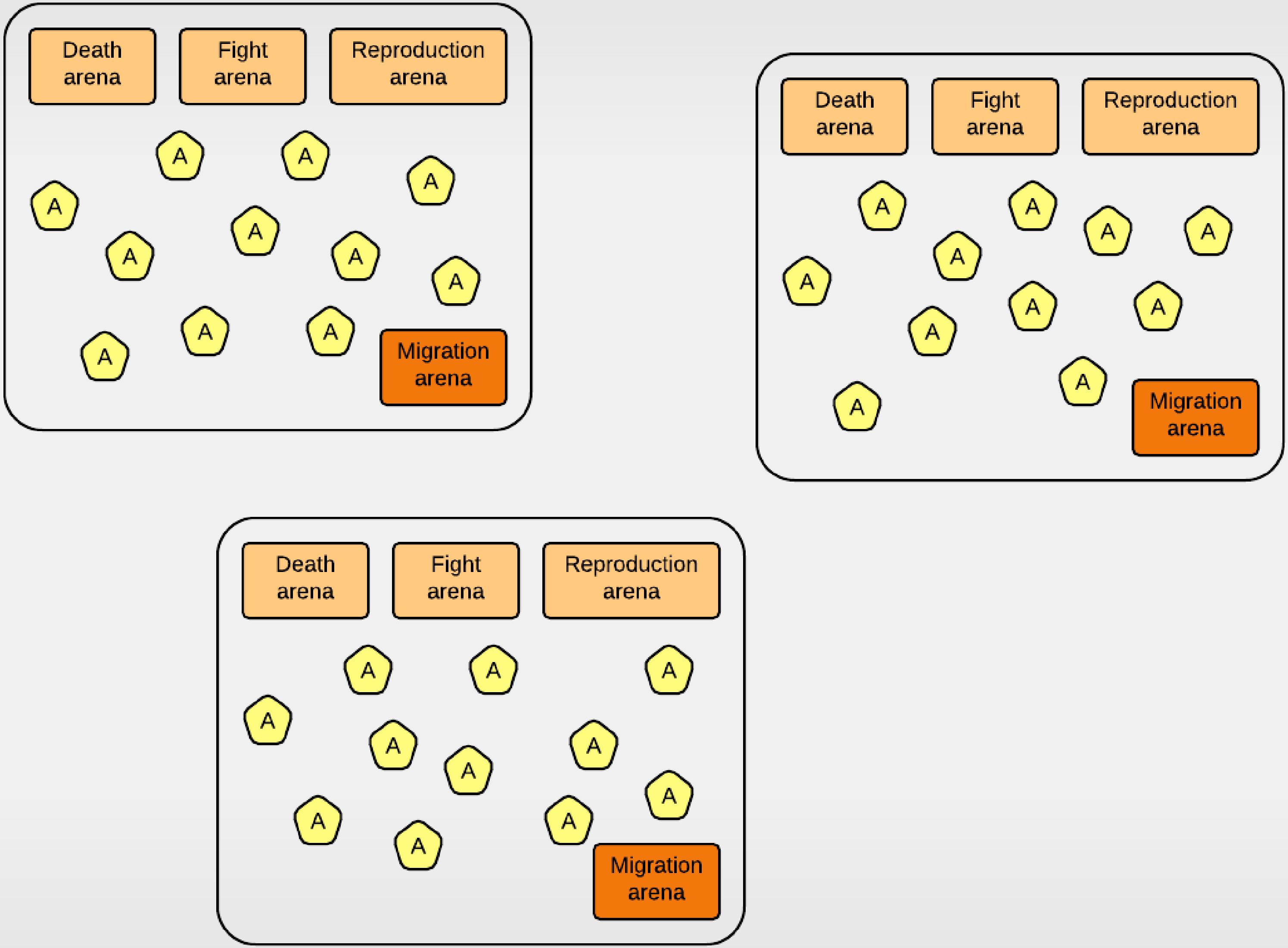
Coarse-grained (skel)

```
ShuffleFun = fun({IsNo, Island}) ->
                      {IsNo, shuffle(Island)}
             end,

Pipe = {pipe, [{seq, TagFun},
               {seq, GroupFun},
               {seq, MigrateFun},
               {seq, WorkFun},
               {seq, ShuffleFun}]},

Result = skel:do([{map, [{feedback,
                          [Pipe],
                          _While = fun(_Islands) ->
                                           os:timestamp() < EndTime
                                   end}],
                   Workers}],
                 [Population]),
```

Fine-grained

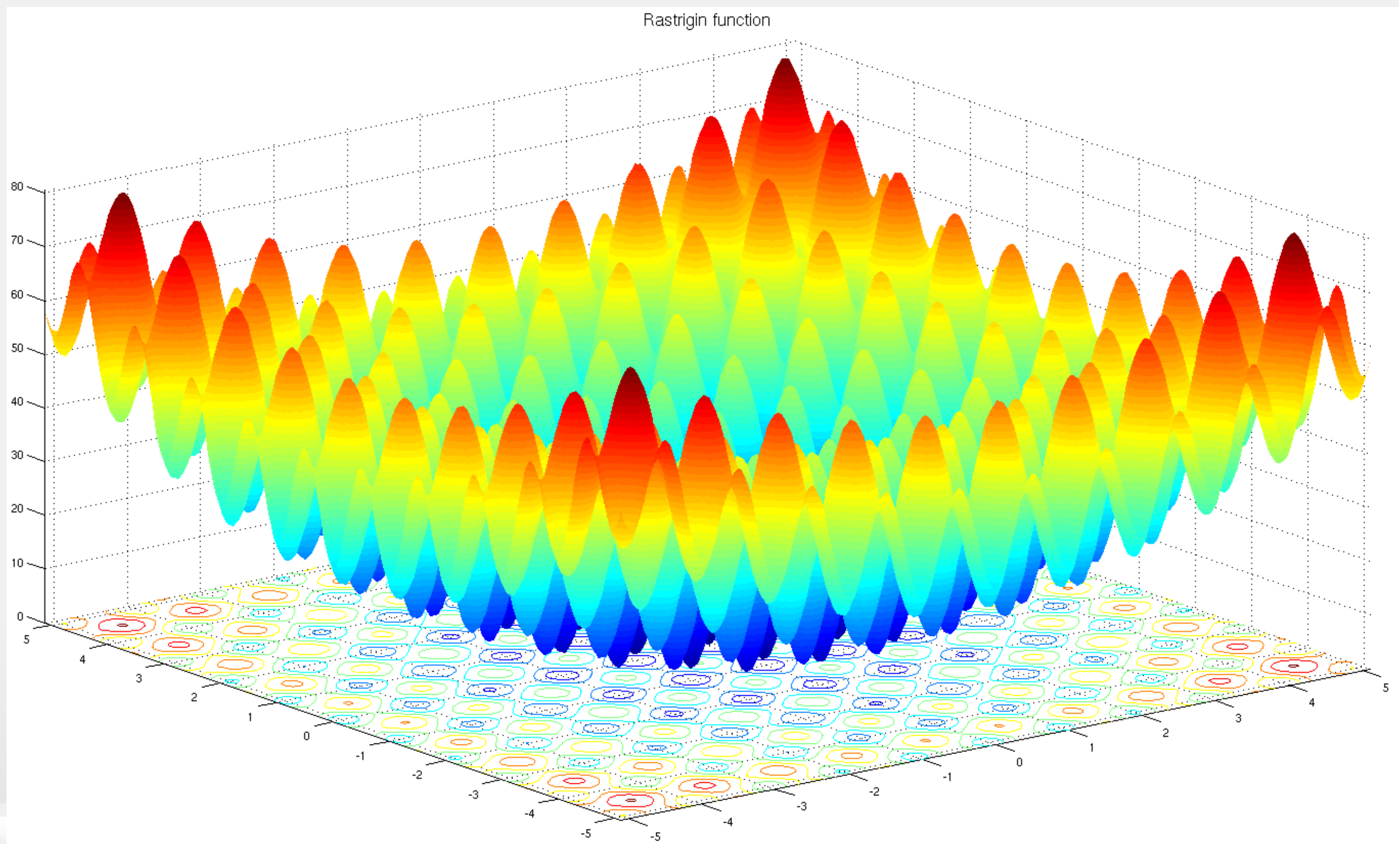


Scaling

Optimized function

- Optimization of Rastrigin function
- 100 dimensions
- 64 islands

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right]$$



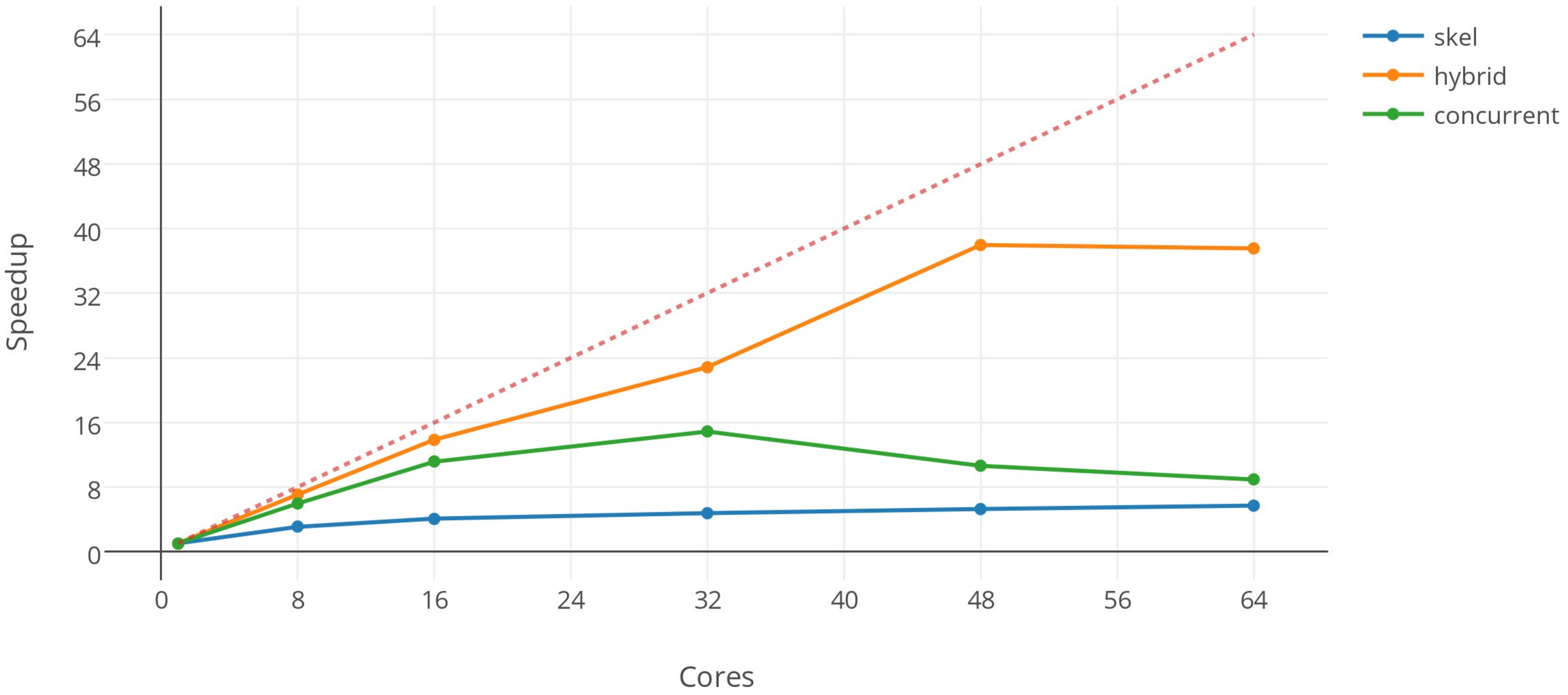
Test infrastructure

- Academic Computer Centre CYFRONET AGH Kraków, Poland
- Testing machine:
  - 1 hardware node
  - Processor: AMD Opteron 6276 2,3 GHz
  - Number of CPUs: 4
  - Total cores: 64
  - RAM: 256 GB
  - OS: Scientific Linux

Initial approach

- N cores, N times faster – not that simple!
- Up to 8 cores Erlang scales well
- Problems arise when the application is run on a higher number of cores

Initial approach



Diagnosing issues

- Not a typical Erlang use case
- Standard profilers (eprof, percept, bencherl)
- Checking schedulers load
  - `erlang:statistics(scheduler_wall_time)`
- lcnt (http://www.erlang.org/doc/man/lcnt.html)
  - Runtime system lock profiling tool
  - Measures lock contention
  - Requires VM recompilation (`--enable-lock-counter`)
  - Introduces some overhead

lcnt output

32 CORES:

lock	id	#tries	#collisions	collisions [%]	time [us]	duration [%]
make_ref	1	13247682	3057885	23.0824	14192658	22.0958
timeofday	1	6757268	1318938	19.5188	7911404	12.3168
run_queue	64	74262017	847072	1.1407	3879060	6.0391
pix_lock	1024	29215	17	0.0582	1390432	2.1647

48 CORES:

make_ref	1	7899140	1209364	15.3101	37203715	54.5142
pix_lock	1024	127348	192	0.1508	8623504	12.6359
timeofday	1	3953082	686742	17.3723	7072442	10.3632
run_queue	64	44374363	305127	0.6876	3094534	4.5344

64 CORES:

make_ref	1	8257415	2978439	36.0699	775283090	1134.5281
pix_lock	1024	632012	896	0.1418	31974180	46.7901
timeofday	1	4203173	631989	15.0360	5324563	7.7918
run_queue	64	47934713	316483	0.6602	2276438	3.3313

lcnt output

32 CORES:

lock	id	#tries	#collisions	collisions [%]	time [us]	duration [%]
make_ref	**1**	**13247682**	**3057885**	**23.0824**	**14192658**	**22.0958**
timeofday	1	6757268	1318938	19.5188	7911404	12.3168
run_queue	64	74262017	847072	1.1407	3879060	6.0391
pix_lock	1024	29215	17	0.0582	1390432	2.1647

48 CORES:

make_ref	**1**	**7899140**	**1209364**	**15.3101**	**37203715**	**54.5142**
pix_lock	1024	127348	192	0.1508	8623504	12.6359
timeofday	1	3953082	686742	17.3723	7072442	10.3632
run_queue	64	44374363	305127	0.6876	3094534	4.5344

64 CORES:

make_ref	**1**	**8257415**	**2978439**	**36.0699**	**775283090**	**1134.5281**
pix_lock	1024	632012	896	0.1418	31974180	46.7901
timeofday	1	4203173	631989	15.0360	5324563	7.7918
run_queue	64	47934713	316483	0.6602	2276438	3.3313

Removing *make_ref*

- Attempt to get rid of the *make_ref* lock
- Application consists of several *gen_server* processes
- Substitute *gen_server:call* with *gen_server:cast* wherever possible

```
call_arena(Pid, Agent) ->
    gen_server:cast(Pid, {interact, self(), Agent}),
    receive
        {response, Reply} ->
            Reply
    end.
```

lcnt results

32 CORES:

lock	id	#tries	#collisions	collisions [%]	time [us]	duration [%]
timeofday	1	7345211	1666056	22.6822	11071384	17.1615
run_queue	64	75714562	931454	1.2302	5378769	8.3375
timer_wheel	1	3613450	224958	6.2256	1131863	1.7545
pix_lock	1024	22168	85	0.3834	919860	1.4259

48 CORES:

timeofday	1	5299733	1239475	23.3875	13788049	21.6985
pix_lock	1024	78638	46	0.0585	5809275	9.1422
run_queue	64	54858096	459805	0.8382	5007581	7.8805
timer_wheel	1	2385084	112436	4.7141	1000315	1.5742

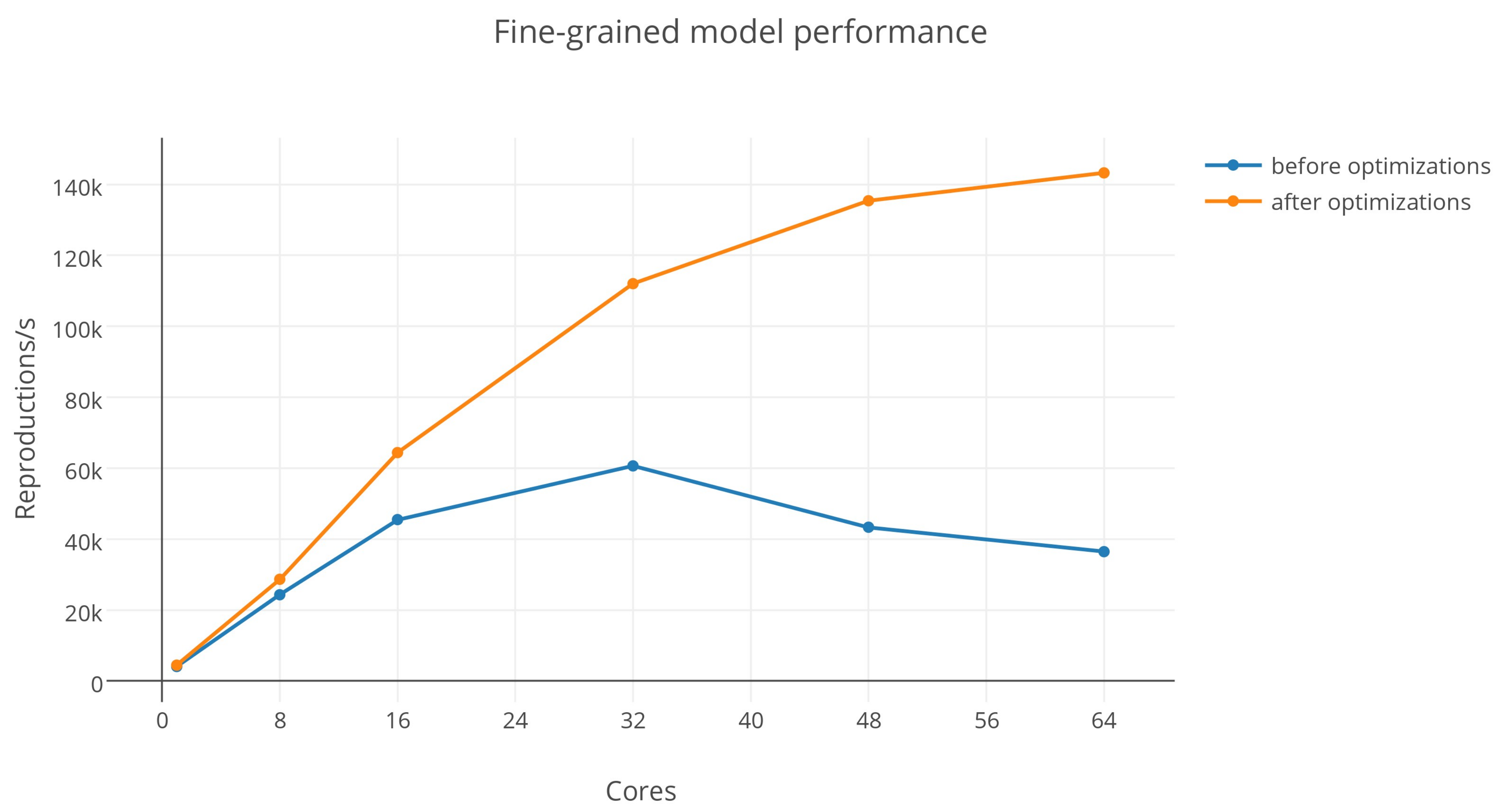
64 CORES:

pix_lock	1024	348272	154	0.0442	21450829	33.2841
timeofday	1	5085784	845736	16.6294	7829344	12.1484
run_queue	64	51053700	310745	0.6087	2571008	3.9893
timer_wheel	1	2317335	79159	3.4159	501694	0.7785

Logger

- Introduced *exometer* in the place of a previous logger `https://github.com/Feuerlabs/exometer`
- Attempt to reduce contentions on the *timeofday* lock
- Built-in counters work well
- Fitness monitoring is <u>expensive</u>. Approaches:
  - Built-in histogram entry
  - ETS probes
  - NIFs

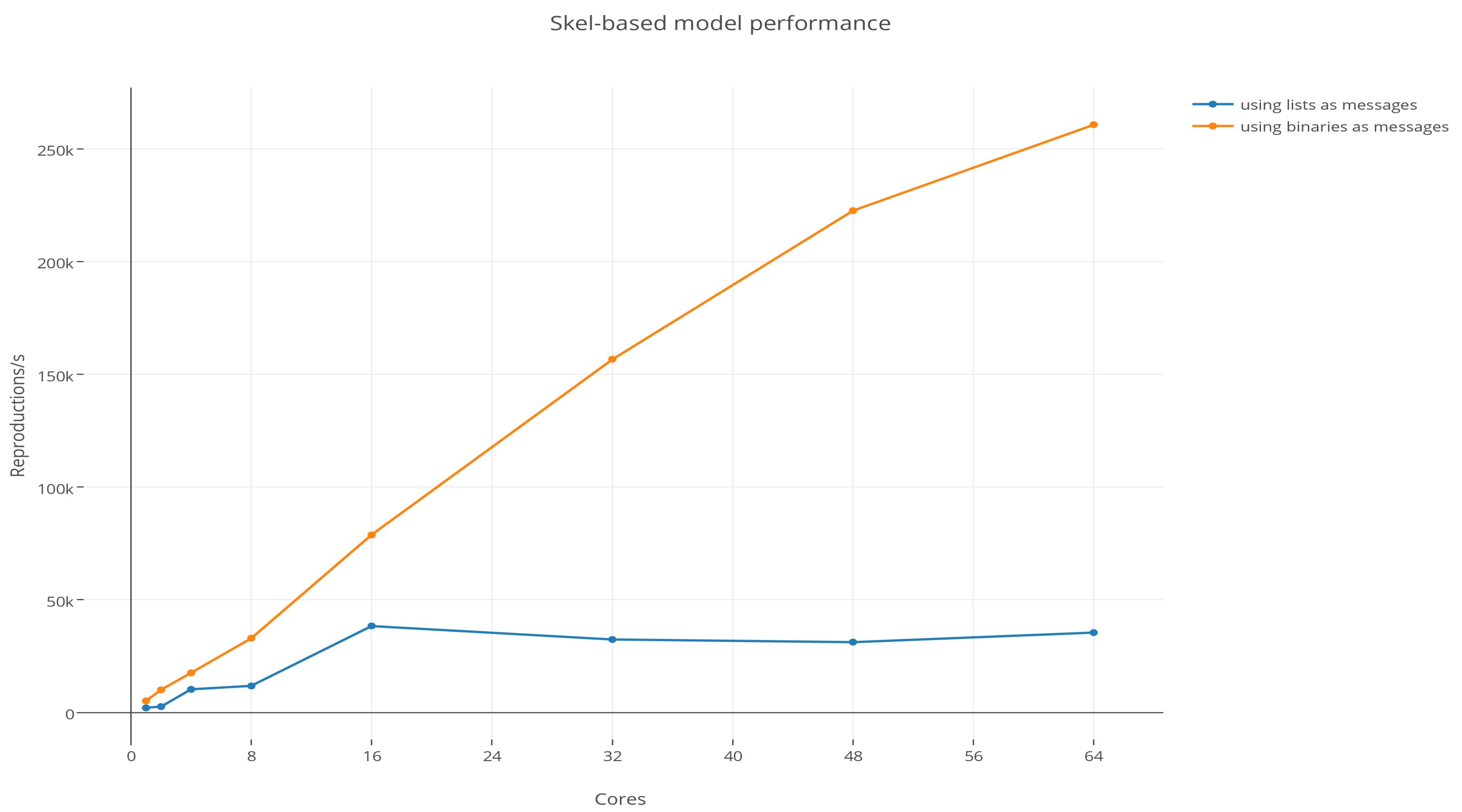
Performance improvement



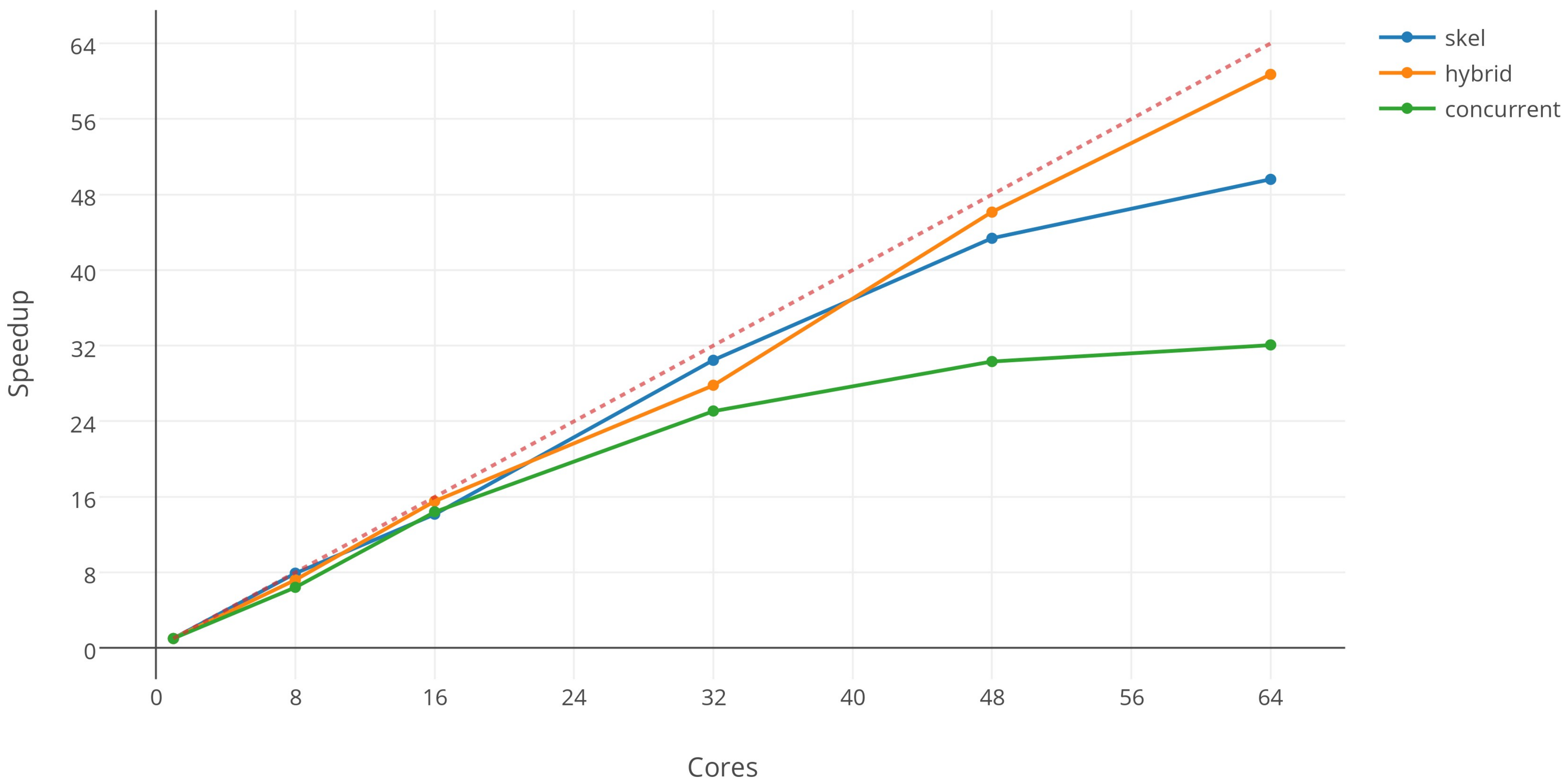
Skel improvement

- Using binary data type instead of Erlang lists
- All data is copied in messages between processes
  - Except for binaries over 64 bytes on the same node
- Cost of decoding/encoding becomes smaller than gain of not copying

Skel improvement results



Final scalability



Summary

- Scaling Erlang vertically is not completely painless
- Asynchronous communication
- Don't check the time/timer too often
- Use binaries instead of lists
- Check your locks (lcnt)
- Use the latest VM
- Ask other people

Links

- http://github.com/ParaPhraseAGH/erlang-emas
- http://github.com/ParaPhraseAGH/erlang-mas
- http://www.paraphrase.agh.edu.pl/
- http://paraphrase-ict.eu/
- http://github.com/ParaPhrase/skel