erlang at hover.in

5 Choices to rule them all

Bhasker V Kode

co-founder & CTO at hover.in

at Commercial Users of Functional Programming 2009,

Edinburgh

September 4th, 2009

http://developers.hover.in

#1. serial vs parallel

"The domino effect is a chain reaction that occurs when a small change causes a similar change nearby, which then will cause another similar change."

via wikipedia page on Domino Effect

small change causes a similar change nearby

similiar change causes another similar change.

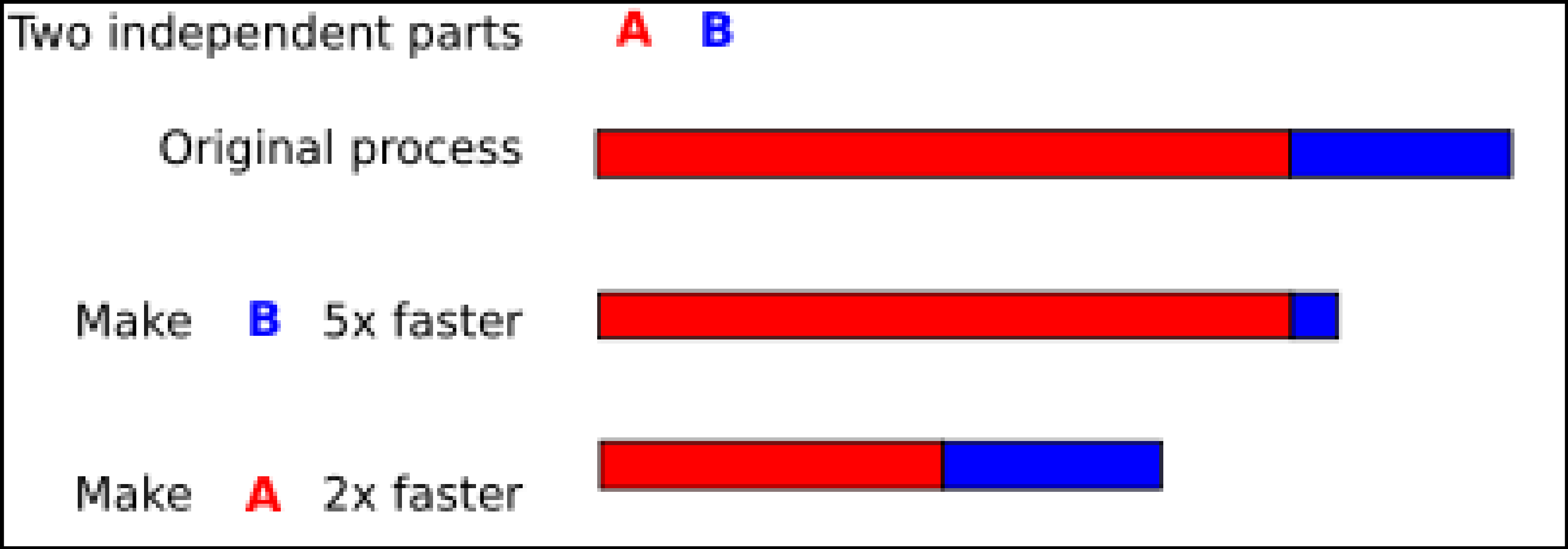
domino effect is a chain reaction caused by a small change.

```
% Or functionally speaking in erlang!
small_change(Changes)->
   similar_change(Changes).

similiar_change([NearBy|Rest])->
  chain_reaction(NearBy),
  similar_change(Rest);

similiar_change( [] ) ->
  “domino effect”.

chain_reaction(NearBy)->
  “wow”.
```

wrt flowcontrol...

- great to handle both bursts or silent traffic & to determine bottlenecks.(eg ur own,rabbitmq,etc)
- **eg1:** when we addjobs to the queue, if it takes greater than X consistently we move it to high traffic bracket, do things differently, possibly add workers **or** ignore based on the task.
- **eg2:** amazon shopping carts, are known to be extra resilient to write failures, (dont mind multiple versions of them over time)

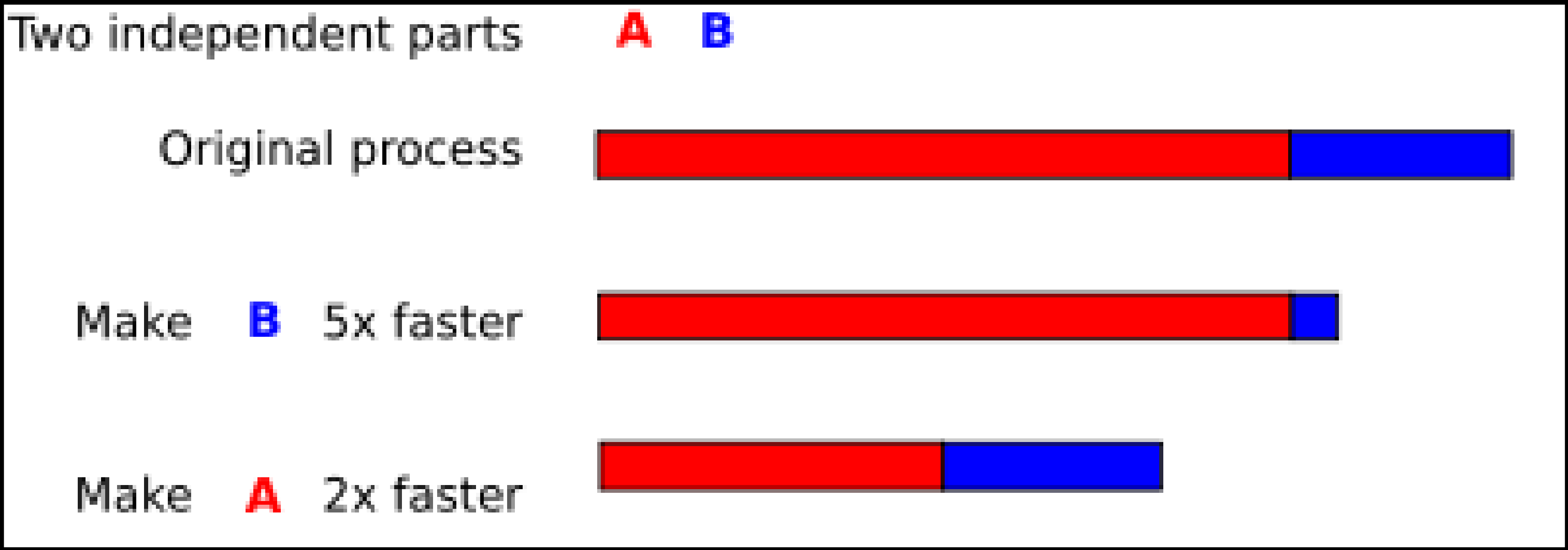
wrt parallel computing...



“the small portions of the program which cannot be parallelized will limit the overall speed-up available ”

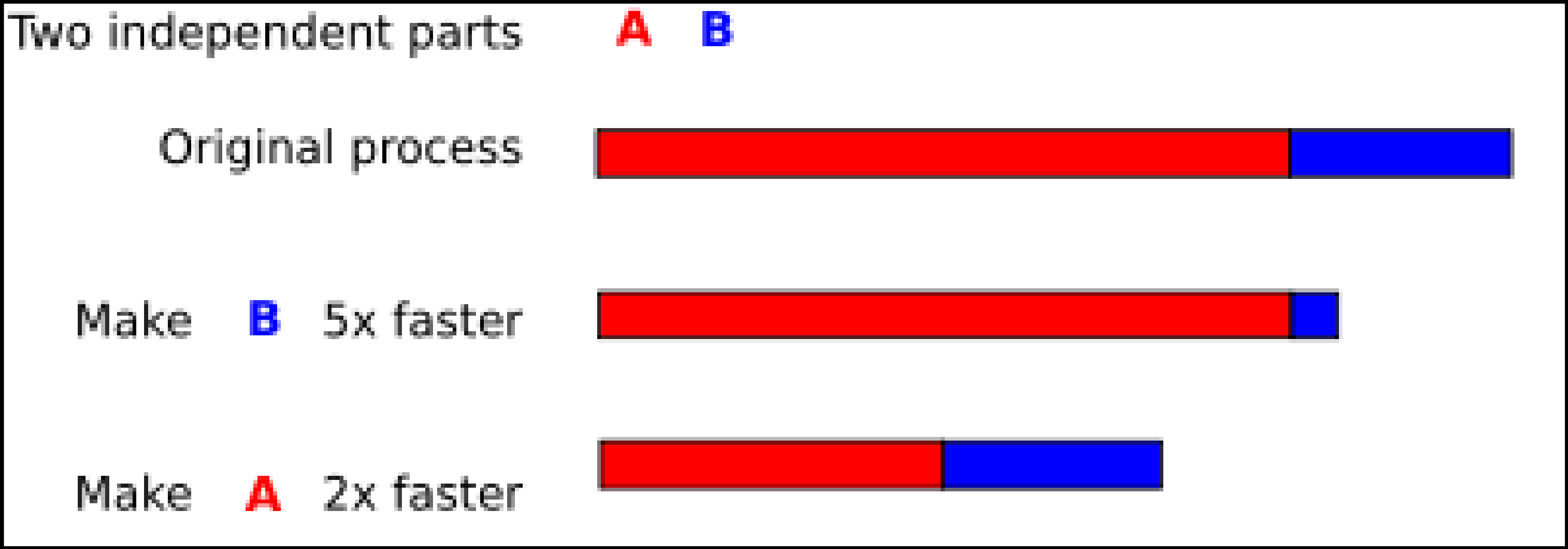
- _____ 's Law ?

wrt parallel computing...



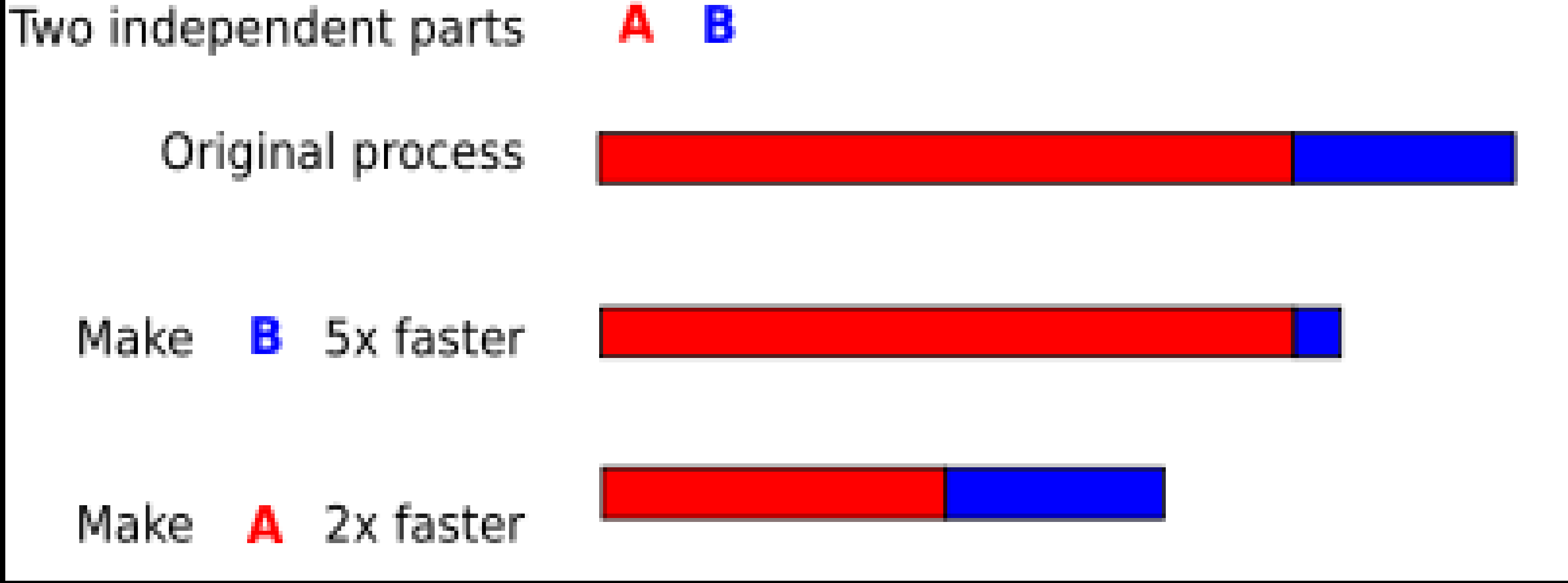
"the small portions of the program which cannot be parallelized will limit the overall speed-up available "

- Amdahl's Law



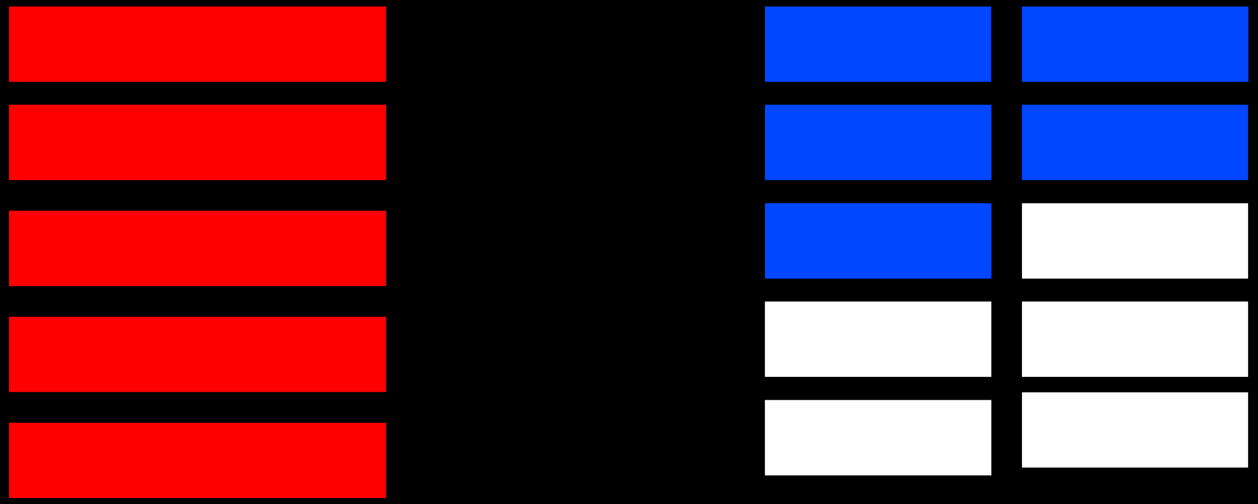
A's and B's that we've faced at hover.in :

- url hit counters **(B)**, priority queue based crawling**(A)**
- writes to create index **(B)**, search's to create inverted index **(A)**
- dumping text files **(B)**, loading them to backend **(A)**
- all ^ shared one common method to boost performance – seperate **flowcontrols for A,B**



Further , A & B need'nt be serial, but a batch of A's & a parallel batch of B's running tasks serially. Flowcontrol implemented by tail-recursive servers handling bursty AND slow traffic well.

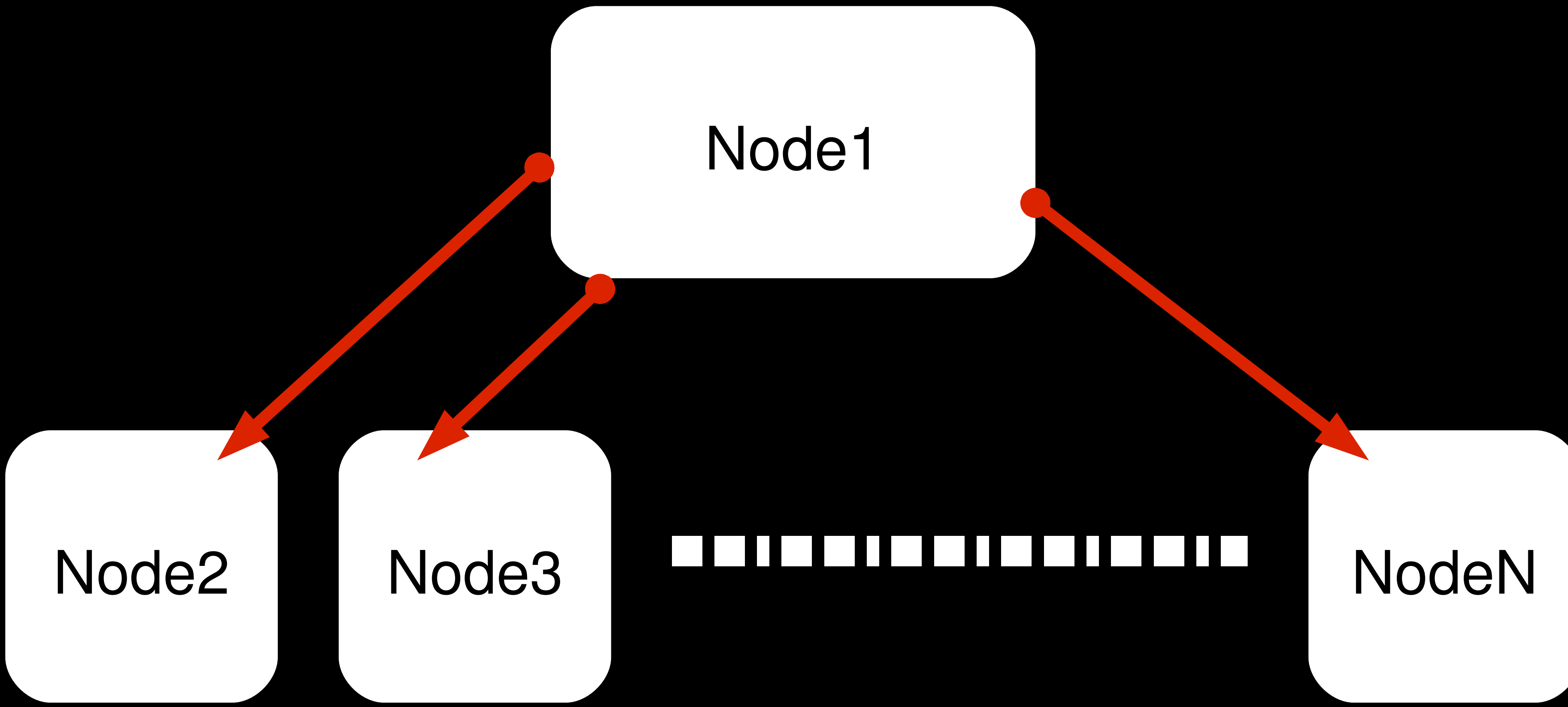
flowcontrol 1 **flowcontrol 2**



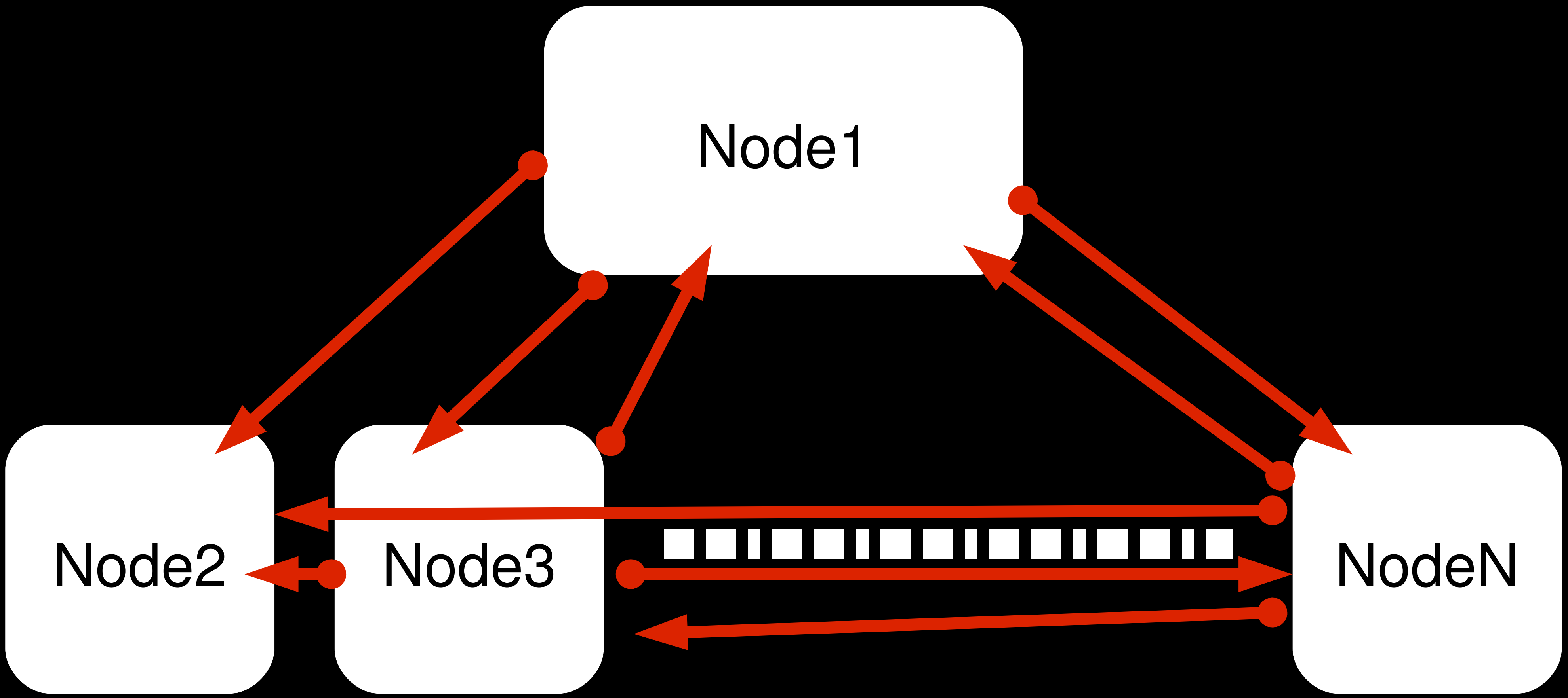
where is free cpu / time for handling **2x** B tasks

#2. distributed vs local

in this uber-cool demo, I “**node1**” split this trivial task to the other 1000 nodes. this will be done in no time. Ha!



in this uber-cool demo, I “**node2**” ALSO split this trivial task to the other 1000 nodes. Ha!



in this uber-cool demo, I “**nodeN**” ALSO split this trivial task to the other 1000 nodes. Ha!

in other words...

- will none of the other N nodes behave as a master?
- won't most your calls be rpc if several nodes try to be masters and ping every other node ?
- would you prefer a distributed non-master setup?
- would you rather load-balance the jobs where each node does what it must do, and does only those jobs (unless failover)
- would you prefer send the task where the data is , rather than one master node accessing all ?
  - **all personal choices at the end of the day...**

how it works at hover.in:

I , "**node X**" will rather do tasks locally since this data is designated to me, rather than rpc'ing all over like crazy !

The meta-data of which node does what calculated by

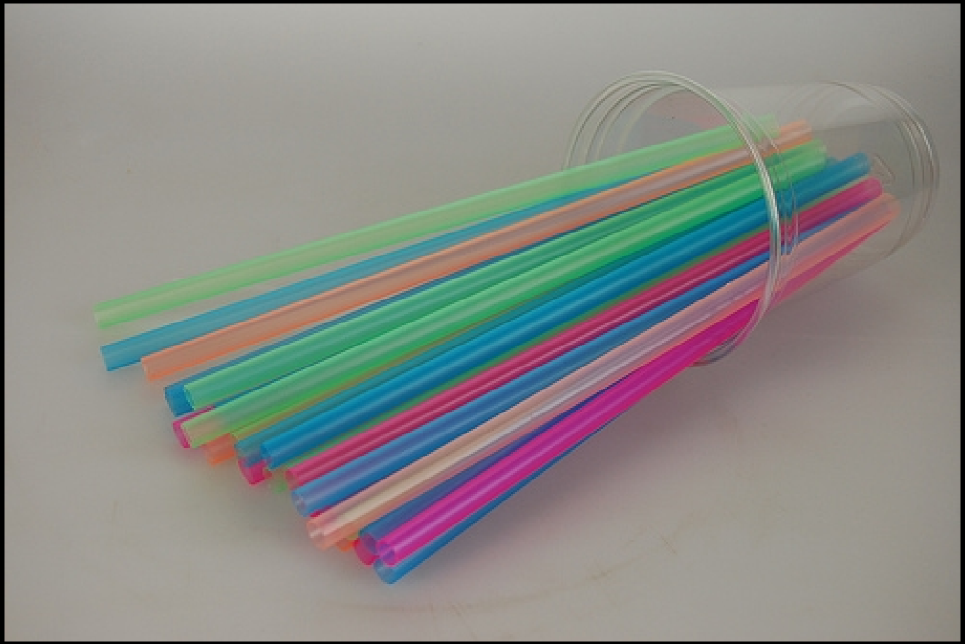
(a) statically assigned *(our choice)* (b) or a hash fn

(c) dynamically reassigned *(maybe later)*

which is made available to nodes by

(a) replication or (b) location transparency *(our choice)*

#3. replication vs location transparency



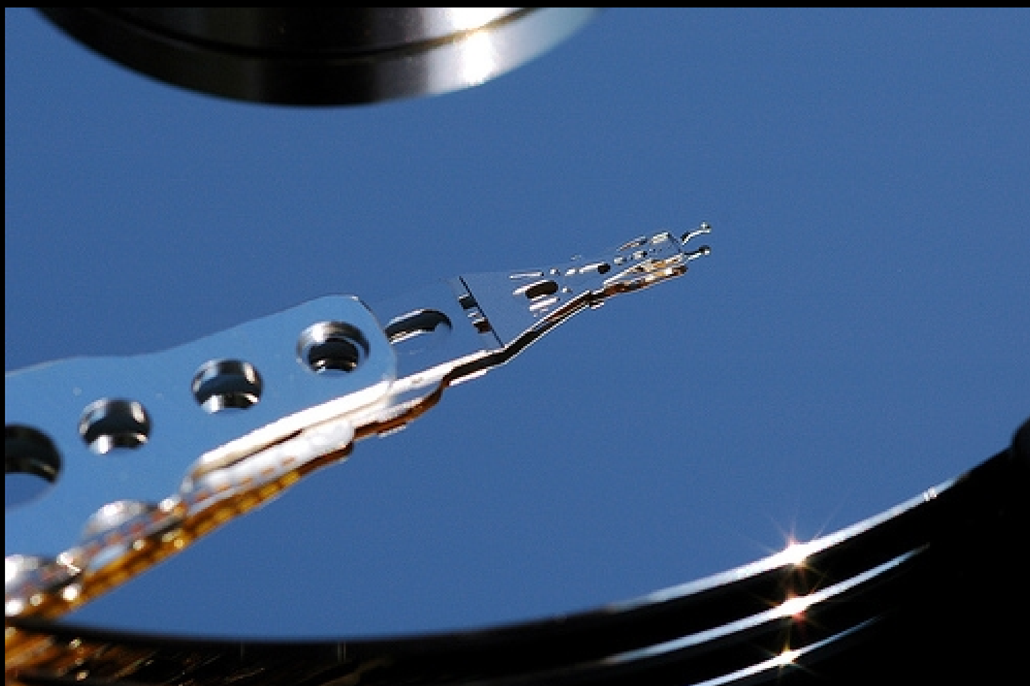
some questons ...

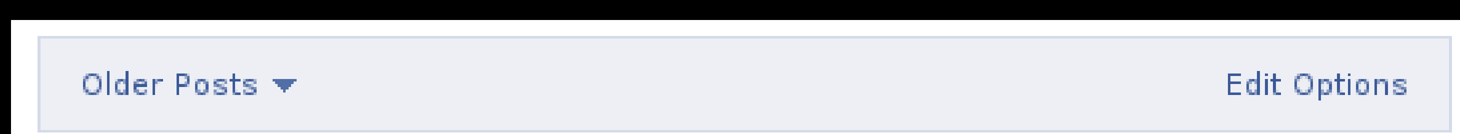
1. replicated on nodes defined by hash function / consistent hashing,etc or statically assigned ?
2. data served from in-memory or completely from disk or a combination of both (LRU cache, etc)?
3. are some instances dedicated readers / writers
4. transactions or no transactions

fortunately erlang/otp/mnesia makes it easy to make highly granular decissions

5. bulk load the data or not (based on your requirements , testing, preferences)
6. run mapreduce / fold over data ? (js in couchdb, or lua with tokyocabinet)

#4. persistent data vs cyclic queues

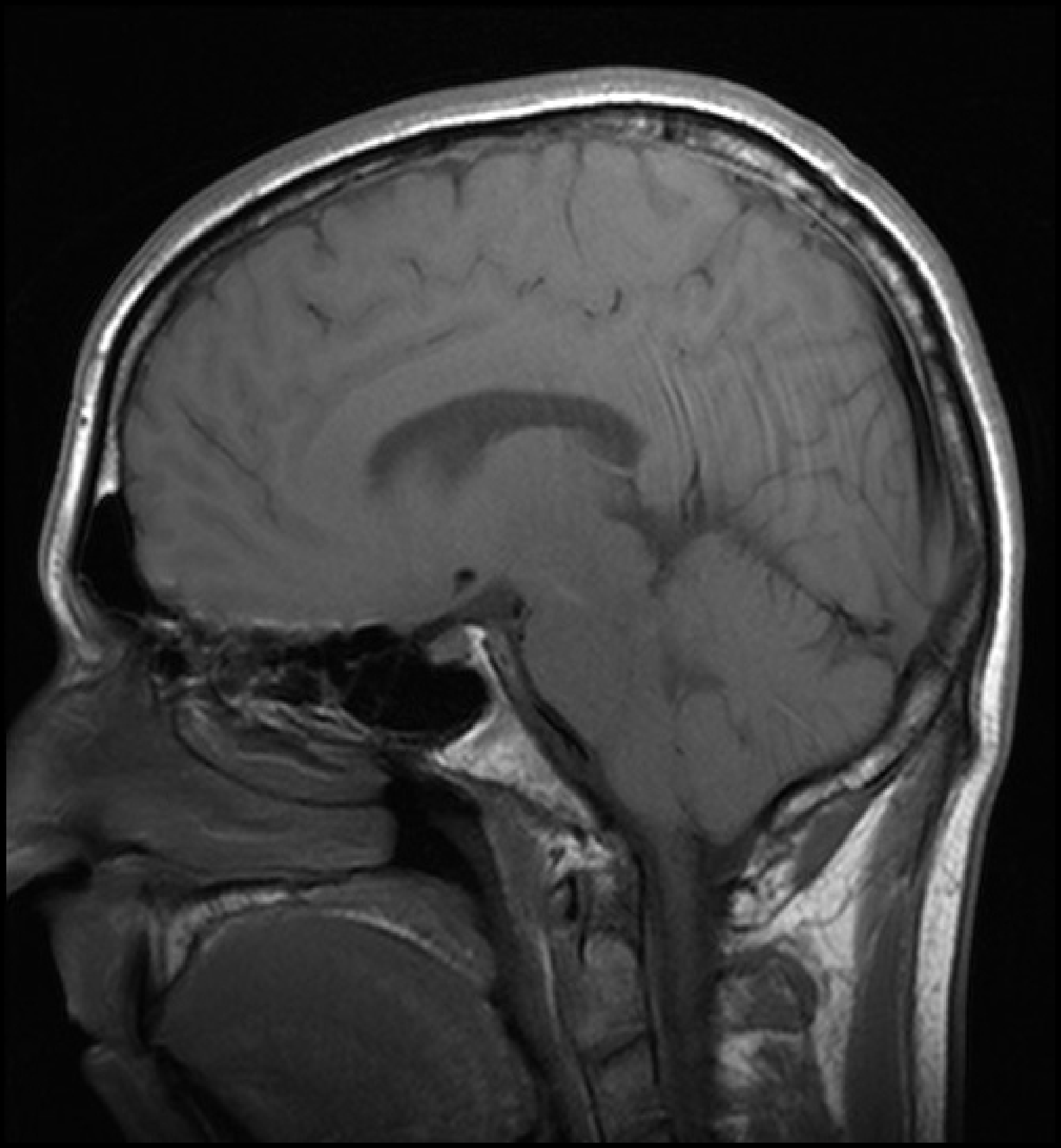




to persist or not to persist...

- **fixed length stores OR round-robin databases OR cyclic queues are an attractive option** ***NEW!***
- great for recission , cutting costs ! just overwrite data
- speedier search's with predictable processing times!
- more realtime, since data flushed based on FIFO
- but risky if you don't have sufficient data
- but pro's mostly outdo cons!
- easy to store/distribute as in-memory data structures
- useful for more buzz-analytics, trend detection, etc that works real-time with less overheads

#5. in-memory vs disk



in-memory is the new embedded

- servers as of '09 typically have 4 - 32 GB RAM
- several companies adding loads of nodes for primarily in-memory operations, caching, etc
- caching systems avoid disk/db , for temporal processing tasks makes sense
- usage of in-memory data structures at hover.in :
  - in-memory caching system , sets
  - LRU cache's, trending topics , debugging, etc

hi_cache_worker

- a circular queue implemented via gen_server
- set (ID , Key , Value , OptionsList)

Options are {purge, <true| false>}

{ size , <integer> }

{ set_callback , <Function> }

{ delete_callback , <Function> }

{ get_callback , <Function> }

{ timeout, <int>, <Function> }

ID is usually a siteid or "global"

- C = hi_cache_worker,
  C:set (User1, “recent_saved” , Value)
  C:set (“global”, “recent_hits” , Value
  [{size,1000}])

  C:get (“global”,”recent_voted”)
  C:get (User1,”recenthits”)
  C:get (User1,”recent_cron_times”)
- (Note: initially used in debugging internally -> then reporting -> next in public community stats)

7 rules of in-memory capacity planning

(1) shard thy data to make it sufficiently un-related

(2) implementing flowcontrol

(3) all data is important, but some less important

(4) time spent x RAM utilization = a constant

(5) before every succesful persistent write & after every succesful persistent read is an in-memory one

(6) know thy RAM, trial/error to find ideal dataload

(7) what cannot be measured cannot be improved

- hover.in founded late 2007
- the web ~ 10- 20 years old
- humans 100's of thousands of years
- but **bacteria**.... around for millions of years

  ... so this talk is going to be about what we can learn from **bacteria**, the **brain**, and **memory** in a concurrent world followed by hover.in's erlang setup and lessons learnt

some traits of bacteria

- each bacteria cell spawns its own proteins
- All bacteria have some sort of some presence & replies associated, *(asynchronous comm.)*
- group dynamics exhibits '*list fold'* ish operation
- only when the **Accumulator** is > some guard clause, will group-dynamics of making light (bioluminiscence) work *(eg: in deep sea)*

supervisors, workers

- as bacteria grow, they split into two. when muscle tears, it knows exactly what to replace.
- erlang supervisors can decide restart policies: if one worker fails, restart all .... or if one worker fails, restart just that worker, more tweaks.
- can spawn multiple workers on the fly, much like the need for launching a new ec2 instant

supervisors, workers

- as bacteria grow, they split into two. when muscle tears, it knows exactly what to replace.
- erlang supervisors can decide restart policies: if one worker fails, restart all .... or if one worker fails, restart just that worker, more tweaks.
- can spawn multiple workers on the fly, much like the need for launching a new ec2 instant

inter-species communication

- if you look at your skin – consists of very many different species, but all bacteria found to communicate using one common chemical language.

inter-species communication

- if you look at your skin – consists of very many different species, but all bacteria found to communicate using one common chemical language.

  hmmmmmmmmmmmmmmmmm.............

  ....serialization ?!

  ....a common protein interpretor ?!

  ....or perhaps just-in-time protein compilation?!

interspecies comm. in practice

- attempts at **serialization** , cross language communication include:
  - **thrift** (by facebook)
  - **protocol buffers** (by google)
  - en/decoding , port based communication (erlang<->python at hover.in)
  - rabbitMQ shows speeds of several thousands of msgs/sec between python <-> erlang (by using...?)

talking about scaling

The brain of the worker honeybee weighs about **1mg** (~ 950,000 neurons)

- Flies acrobatically , recognizes patterns, navigates , communicates, etc
- Energy consumption: 10–15 J/op, at least 106 more efficient than digital silicon neurons

the human brain

- 100 billion neurons, stores ~100 TB
- Differential analysis e.g., we compute color
- Multiple inputs: sight, sound, taste, smell, touch
- Facial recognition subcircuits, peripheral vision
- in essence - the left & right brain vary in:
  left -> persistent disk , handles past/future
  right -> temporal caches! , handles present

summary of tech at hover.in

- LYME stack since ~dec 07 , 4 nodes (64-bit 4GB)
- python crawler, associated NLP parsers, index's now in tokyo cabinet , inverted index's in erlang 's mnesia db,cpu time-splicing algo's for cron's app, priority queue's for heat-seeking algo's app, flowcontrol, caching, pagination apps, remote node debugger, cyclic queue workers, headless-firefox for thumbnails
- touched 1 million hovers/month in May'09 after launching closed beta to publishers in Jan 09

brief introduction to hover.in

choose words from your blog, & decide what content / ad you want when you hover* over it

* or other events like click,right click,etc

or...

the worlds first user-engagement platform for brands via in-text broadcasting

or

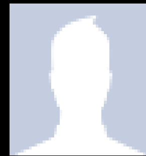
lets web publishers push client-side event handling to the cloud, to run various rich applications called *hoverlets*

demo at http://start.hover.in/ and http://hover.in/demo

more at http://hover.in , http://developers.hover.in/blog/

summary of our erlang modules

rewrites.erl error.erl frag_mnesia.erl hi_api_response.erl hi_appmods_api_user.erl
hi_cache_app.erl , hi_cache_sup.erl hoverlingo.erl hi_cache_worker.erl
hi_lru_worker.erl hi_classes.erl hi_community.erl
hi_cron_hoverletupdater_app.erl hi_cron_hoverletupdater.erl
hi_cron_hoverletupdater_sup.erl hi_cron_kwebucket.erl hi_cron_kweload.erl
hi_crypto.erl hi_daily_stats.erl hi_flowcontrol_hoverletupdater.erl
hi_htmlutils_site.erl hi_hybridq_app.erl hi_hybridq_sup.erl hi_hybridq_worker.erl
hi_login.erl hi_mailer.erl hi_messaging_app.erl hi_messaging_sup.erl
hi_messaging_worker.erl hi_mgr_crawler.erl hi_mgr_db_console.erl
hi_mgr_db.erl hi_mgr_db_mnesia.erl hi_mgr_hoverlet.erl hi_mgr_kw.erl
hi_mgr_node.erl hi_mgr_thumbs.erl hi_mgr_traffic.erl hi_nlp.erl hi_normalizer.erl
hi_pagination_app.erl hi_pagination_sup.erl, hi_pagination_worker.erl
hi_pmap.erl hi_register_app.erl hi_register.erl, hi_register_sup.erl,
hi_register_worker.erl hi_render_hoverlet_worker.erl hi_rrd.erl , hi_rrd_worker.erl
hi_settings.erl hi_sid.erl hi_site.erl hi_stat.erl hi_stats_distribution.erl
hi_stats_overview.erl hi_str.erl hi_trees.erl hi_utf8.erl hi_yaws.erl
& medici src (erlang tokyo cabinet / tyrant client)

thank you

http://developers.hover.in

references

- All images courtesy Creative Commons-licensed content for commercial use, adaptation, modification or building upon from Flickr
- http://erlang.org , wikipedia articles on Parallel computing
- amazing brain-related talks at http://ted.com ,
- go read more about the brain, and hack on erlang NOW!
- shoutout to everyone at #erlang !
- get in touch with us on our dev blog http://developers.hover.in , on twitter @hoverin, or mail me at kode at hover dot in.