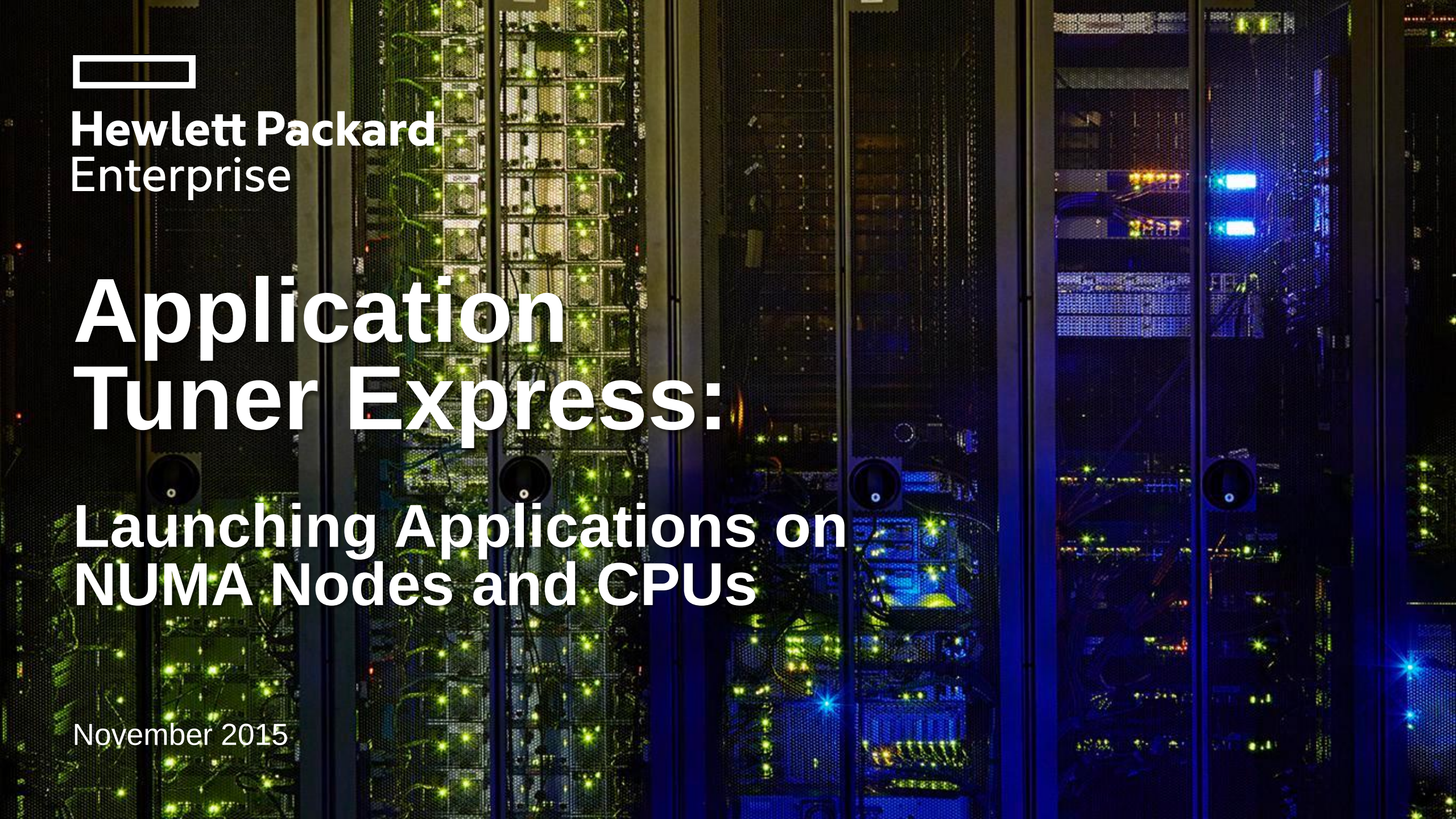


Hewlett Packard Enterprise

Application Tuner Express:

Launching Applications on NUMA Nodes and CPUs

November 2015

Application Tuner Express overview

– ATX is a utility that will make NUMA unaware applications more NUMA aware
  – No application changes are needed!

– ATX controls the distribution of an application's processes and threads in a NUMA environment
  – Several different NUMA node and CPU launch policies are provided to obtain an optimal distribution.

– ATX is similar to the Linux `numactl` command:
  – `numactl` will constrain an application to a set of nodes
  – ATX will distribute an application around a set of nodes

– Benefit of ATX varies by platform and application
  – Higher socket count platforms benefit more than lower socket count platforms
  – NUMA-aware applications benefit less than applications without NUMA awareness

ATX Process and Thread Launch Policies

- Launch Policies control how an application's processes and threads are distributed among the NUMA nodes on the system
- There is both a process launch policy and a thread launch policy for a process.
- These two policies do not have to be the same in a given process.
  - Process launch policies govern the NUMA binding of a child process created from `fork()`, `vfork()`, etc.
  - Thread launch policies govern the NUMA binding of a sibling thread created from `pthread_create()`, `clone()`, etc.
- Launch policies are inherited by the created process or thread.
- Launch policies are also inherited across an exec call to start a new executable.

ATX Launch Policy types

There are 4 launch policy types (available for both processes and threads):

– **Round-Robin (RR):** Each time a process (or thread) is created it will be bound to the next NUMA node in the list of available nodes. This ensures even distribution across all nodes

– **Fill-First (FF):** Each time a process (or thread) is created it will be bound to the same NUMA node until we have created the same number of processes (or threads) in the node as there are CPUs in that node. Once <ncpu> processes have been created future creation will take place in the next NUMA node.

– **Packed (Pack):** All child processes or sibling threads will be bound to the same NUMA node

– **None:** No launch policy is defined. Any child process or sibling thread that is created will inherit any NUMA binding constraints from it's creator.

Process Launch Trees

– The process launch tree is the set of processes that the process launch policy should be applied to.

– The initial process started by ATX (as specified by the user) forms the root of a process launch tree.

– There are both tree-based and flat-based launch trees.

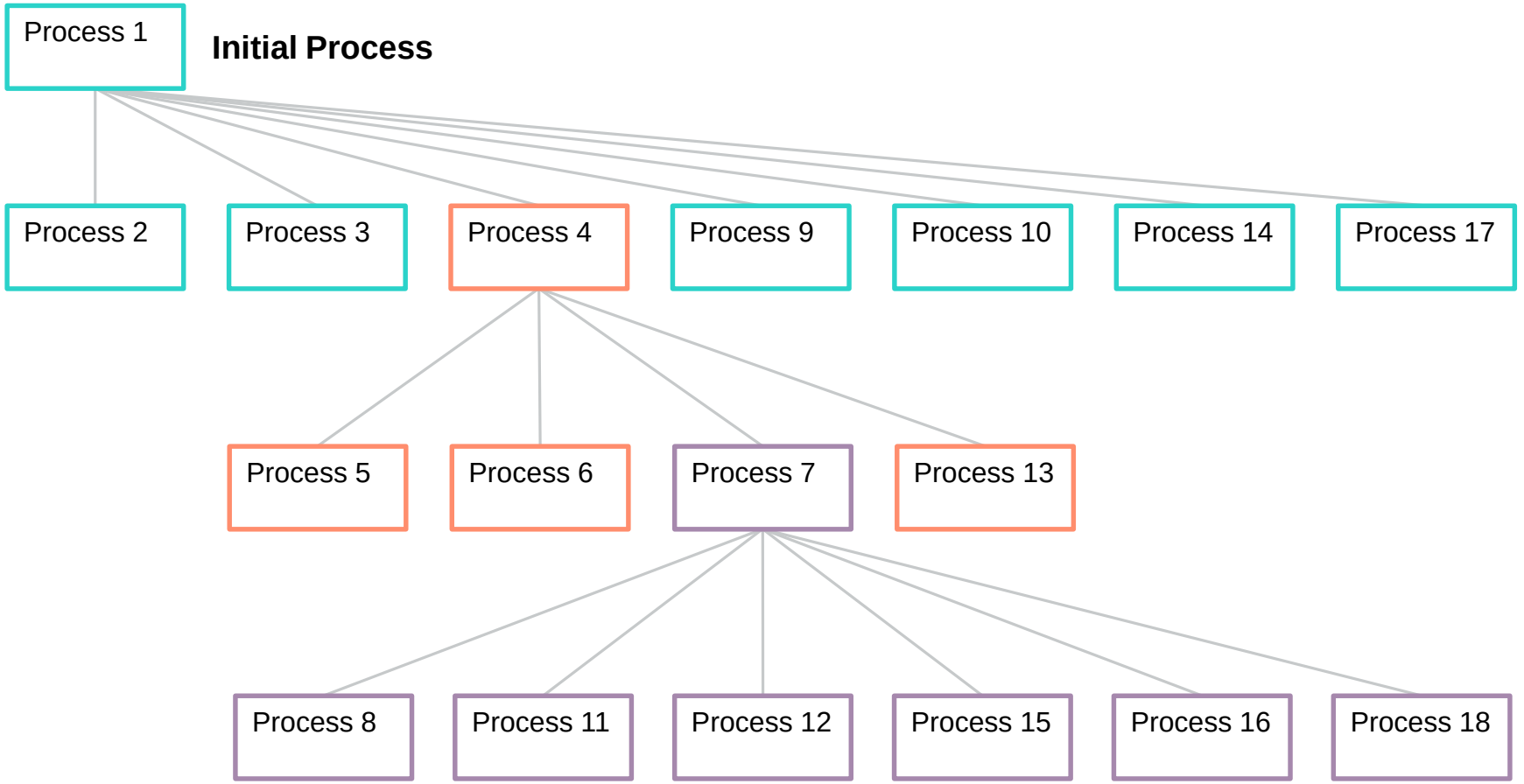
– With a tree-based policy all processes created by the initial process or any of it's descendants, regardless of how deep the parent/child tree is, are in the same launch tree. All of these processes will be launched relative to one another in the order they are created according to the launch policy specified.

– With a flat-based policy the initial process and its direct children form a launch tree. The initial process and it's direct children will be launched relative to one another in the order they are created according to the launch policy specified.

  If one of the children (child A) creates another process then child A becomes the root of a new process launch tree. All of the direct children of child A will be launched relative to one another in the order they are created according to the launch policy specified.

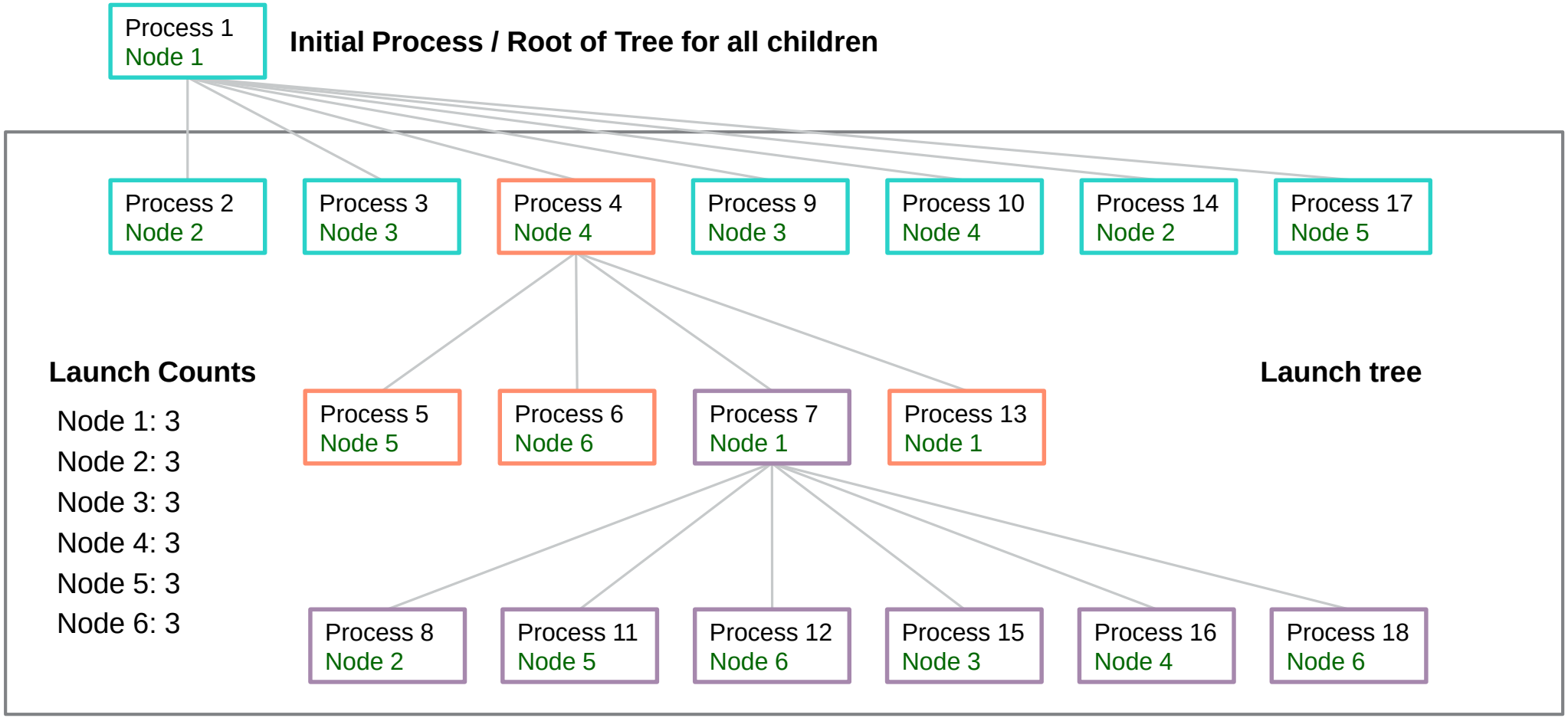
Launch Policy: Sample parent/child tree

Processes are created in the order specified



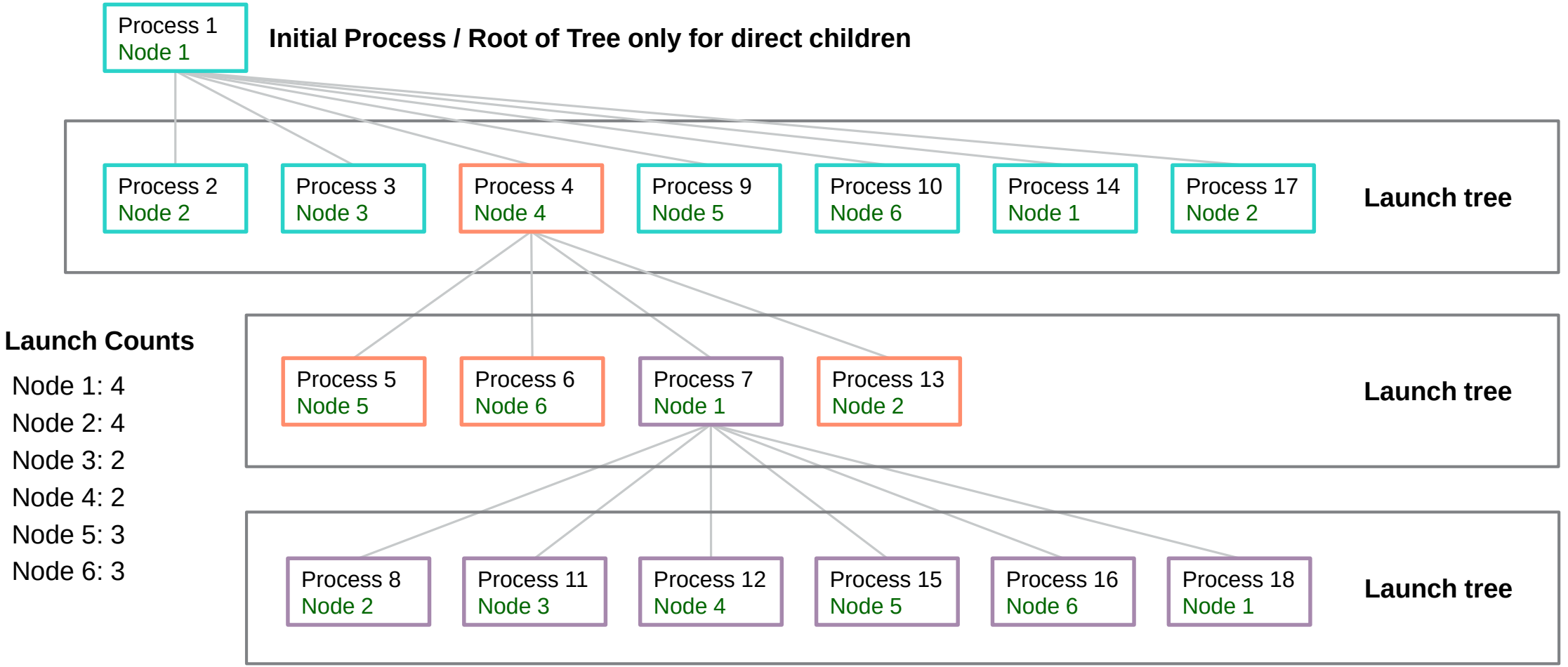
Launch Policy: RR Tree example

Nodes 1-6 available. Processes are created in the order specified. Node # is the resulting node binding



Launch Policy: RR Flat example

Nodes 1-6 available. Processes are created in the order specified. Node # is the resulting node binding



Launch Policies and the Initial Thread

- When a process is created it is governed by the process launch policy. This policy is applied to the initial/main thread of the process.
- When the process creates it's first thread (really the second thread since the initial/main thread of the process was the first) the thread launch policy of the process takes effect for thread creation.
- However, the initial/main thread already has a launch node from the process launch policy. This launch node is not changed.
  - Instead, this node is used as the starting node to implement the thread launch policies for all threads created by this process.
  - If there is no thread policy defined the created threads will inherit any launch node from their creating thread
    - If there is a process policy but no thread policy, all threads in a process have the same launch node. In this case the thread policy essentially defaults to the **pack** policy.

Thread Launch Policies: Tree vs Flat

– A flat-based thread policy applies only to the threads inside a process. All threads of a process are launched relative to one another according to the thread launch policy. Each process manages its own flat-based thread launch tree.

– A tree-based thread policy applies to all threads created by the initial process and any of its descendants, regardless of how deep the parent/child tree is. All threads are launched relative to one another according to the thread launch policy.

CPU Launch Option

- ATX also has a CPU Launch option
- You must still specify a process or thread NUMA launch policy
- Once the launch node has been selected ATX will then select a specific CPU within that node to launch the process or thread
  - CPU selection is round-robin within the node.

ATX Semantics

hpe-atx [*options*] [--] command [*command arguments ...*]

Options are:

-p <*policy*>, --process=<*policy*>
Set the process launch policy to <*policy*>.
<*policy*> can be one of:
rr_tree rr_flat ff_tree ff_flat pack
none *(the default if not specified)*

-t <*policy*>, --thread=<*policy*>
Set the thread launch policy to <*policy*>.
<*policy*> can be one of:
rr_tree rr_flat ff_tree ff_flat pack
none *(the default if not specified)*

-c, --cpu
Also apply the CPU launch policy after determining the
launch node based on the process or thread launch policy

-n <*node-list*>, --nodes=<*node-list*>
Only use the nodes specified in <*node-list*> which is
in the format of "2-4", "1,3,5", or "1,3,6-8‟ (same format as numactl)
(the default is the current node affinity)

-l <*log-file*>, --log=<*log-file*>
Enable logging of process and thread creation *(the default is no logging)*

-e <error-file>, --error=<error-file>
Write any errors encountered to this file also

-r, --remove-data-files
Remove any old leftover data files from previous invocations

-w, --write-by-other
Log and data files will be created with writer permission by “others”

Hewlett Packard Enterprise

Initial Launch Nodes Option

– ATX will adhere to any previous NUMA node affinity through `numactl` or cpusets.

– By default ATX will use all nodes available from its node affinity.

– `command` may be constrained to a smaller list of NUMA nodes by using the `-n <node-list>` or `--nodes=<node-list>` option.

– `node-list` has the same format as a node list for the `numactl` command:

  – `"1"` or `"1,2,3"` or `"1-3"` or `"1,3-5"` or `"1-3,5-7"`

  – A relative `node-list` may be specified as +A,B,C or +A-B and so forth. The `+` indicates that the node numbers are relative to the process' set of allowed nodes.

  – An inverse `node-list` can be specified as !A-C or !A,B,C and so forth. The `!` indicates that all allowed nodes in the current cpuset except these nodes should be used.

  – If the keyword `all` is specified rather than a list of nodes it means use all the nodes in the current cpuset.

Log File Option

– A `log-file` can be specified to record launch events by using the `-l <log-file>` or `--log=<log-file>` option.

– The following events are logged:
  – Process creation – logged by the parent process
  – Process startup – logged by the child process
  – Process exec – one of the exec* APIs was called
  – normal process termination through the exit APIs (implicitly or explicitly)
  – thread creation – logged by the creating thread
  – Thread creation – logged by the newly created thread

– Abnormal process termination events (e.g., killed by a signal) are not logged

– Thread termination events are not logged

Log File Sample

Timestamp	Entry#	TID	PID	PPID	Node	CPU	Log Message	cmdline
0.000155	1	48036	48036	47702	1	76	initial exec start	tst/sjn_test
0.000680	2	48036	48036	47702	1	76	Created PID 48039	tst/sjn_test
0.000902	3	48039	48039	48036	2	30	child start in fork()	tst/sjn_test
0.001131	4	48039	48039	48036	2	30	exit()	tst/sjn_test
1.001081	5	48036	48036	47702	1	76	Created PID 48040	tst/sjn_test
1.001306	6	48040	48040	48036	3	45	child start in fork()	tst/sjn_test
1.001539	7	48040	48040	48036	3	45	_exit()	tst/sjn_test
2.001390	8	48036	48036	47702	1	76	Created PID 48041	tst/sjn_test
2.001586	9	48041	48041	48036	1	77	child start in fork()	tst/sjn_test
2.001804	10	48041	48041	48036	1	77	_Exit()	tst/sjn_test
3.001791	11	48036	48036	47702	1	76	Created PID 48042	tst/sjn_test
3.001996	12	48042	48042	48036	2	31	child start in fork()	tst/sjn_test
3.002241	13	48042	48042	48036	2	31	_exit()	tst/sjn_test
4.002100	14	48036	48036	47702	1	76	Created PID 48043	tst/sjn_test
4.002300	15	48043	48043	48036	3	46	child start in fork()	tst/sjn_test
4.002519	16	48043	48043	48036	3	46	_exit()	tst/sjn_test
5.002491	17	48036	48036	47702	1	76	Created PID 48044	tst/sjn_test
5.002683	18	48044	48044	48036	1	78	child start in fork()	tst/sjn_test
5.002903	19	48044	48044	48036	1	78	_exit()	tst/sjn_test

Remove Data Files Option

– ATX internally maintains an internal data file that is used to keep track of which Node/CPU should be used for the next process or thread launch.

– It is possible that ATX could leave old unused data files around.

– By default, when ATX starts it looks for and removes any old unused internal data files from a previous invocation of ATX.
  – While these data files do not claim a lot of space you can use the `-r` or `--remove-data-files` option by itself to remove any old unused internal data files.
  – For example:

```
hpe-atx -r  or  hpe-atx -remove-data-files
```

– This option does not provide output.

– If unused data files are found they are silently removed.

Write-by-Other Option

- By default ATX will create the <log-file> as well as an internally maintained data file with read/write permissions for owner / group and read permission for other.
  - 0664 or `-rw-rw-r--`
- Some applications may change their user or group ID causing access permission problems for the *<log-file>* and the internal data file.
- The `--write-by-other` or `-w` option will cause the *<log-file>* and the internal data file to be created with read/write permissions for owner, group, and other.
  - 0666 or `-rw-rw-rw-`
- Note: the file-creation mode mask inherited from the parent process or set with the shell's `umask` command may cause files to be created with more restrictive permissions.
  - To ensure ATX creates files with the permissions specified above set the umask with the command:

```
umask 0
```

Error File Option

– If any errors are encountered by ATX the error will be written to `stderr`.
  – Run-time errors (not process startup errors) will also be written to the `log-file` if logging was specified.
– Some applications close `stdout` and `stderr` which can make error reporting difficult.
– The `-e <error-file>` or `--error=<error-file>` option will cause ATX to additionally write any error it encounters to the file `error-file`.
– This file will only be created if ATX encounters an error launching `command` or any of its children.
– It is recommended to always use this option unless you know for certain that `command` does not close `stdout` and `stderr`.

Error Handling

– If ATX encounters an error while launching processes and threads it has three ways of handling the error:

  – Errors that occur before or during the launch of `command` are treated as fatal startup errors and ATX will terminate. These type of errors are usually errors related to bad parameters being passed to ATX.

  – Some errors are survivable errors that do not need to stop ATX or `command` from continuing. An example of a survivable error is if writing to the log file fails then logging is disabled, however, `command` and it's children will continue to run.

  – After `command` has started, if ATX encounters an error it will treat the error as a disable error. ATX will disable itself in the process that encountered the error. All of the initial launch nodes will be used for that process as well as any processes and threads it creates.

– This approach for surviving with reduced functionality and/or disabling when an error is encountered allows the user to cleanly shutdown `command` at an opportune time.

Constraints

– ATX is not currently supported for use within a virtual guest environment

– ATX does not support `32-bit, setuid` or `setgid` executables
  – If encountered ATX will disable itself for that executable and any processes or threads it creates.
  – The executable (and all of it's children) will inherit its launch node or cpu from its creating process
  – The executable is still allowed to run
  – If a <*log-file*> was specified a log entry will be created to notify that this type of executable was encountered

– ATX does not recognize and reconfigure itself as cpus, or all cpus within a node, are placed in an online or offline state.
  – If encountered ATX will disable itself in that process as well as any processes or threads it creates.
  – The process will be allowed to use all of the initial launch nodes
  – The process will continue to run

Examples

- **hpe-atx -p rr_tree -- app args**
  - Run `app` with arguments `args` using the `rr_tree` process launch policy
- **hpe-atx -p rr_tree -t ff_flat app args**
  - Run `app` with arguments `args` using the `rr_tree` process launch policy and the `ff_flat` thread launch policy
- **hpe-atx -p rr_flat -c app args**
  - Run `app` with arguments `args` using the `rr_flat` process launch policy as well as a CPU launching within the node
- **hpe-atx -p ff_tree -l log.txt app args**
  - Run `app` with arguments `args` using the `rr_tree` process launch policy and log launch operations to the file `log.txt.`
- **hpe-atx -p ff_flat -n 1-3 app args**
  - Run `app` with arguments `args` using the `ff_flat` process launch policy and use nodes `1-3` to launch the process and threads created by `app.`
- **hpe-atx -p rr_flat -e errors.txt app args**
  - Run `app` with arguments `args` using the `rr_flat` process launch policy. Log any errors encountered by `hpe-atx` to the file `errors.txt`.

Tuning thoughts…

- Run `hpe-atx -l my_log.txt <cmd>` (no launch policy specified) and look at the resulting log file to understand the entire process tree, parent/child relationships, creation order, etc.
- For applications where there is only one executable to launch it's pretty simple – just try all the launch policies (with and without CPU lunching).
- For applications where there are multiple executables to be started you might want to look at each executable separately.
  - It might be that each executable needs a different launch policy
  - It also might be that only some executables in an application need a launch policy

Hewlett Packard
Enterprise

Thank you