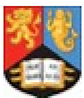
The VM way

- Code is treated as being insecure and being compiled and run in a secure environment (ie: the JVM).
- Runtime execution permissions and an awful set of complicated overhead added to the program.
- Runs effectively slow!

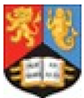
The low tech way

- Code is treated as being secure and has direct access to memory.
- No constraints except those inherited from the machine itself.
- Runs effectively fast!



```
sighndlr(int dummy) {
   ...
   free(global_ptr1);
   free(global_ptr2);
   ...
}
int main(void) {
...
   global_ptr1 = calloc(..., sizeof(...));
   global_ptr2 = calloc(..., sizeof(...));
...
   signal(SIGTERM, sighndlr);
   signal(SIGHUP, sighndlr);
...
}
```

- The signal handler attaches two signals to the same signal handler. Subsequent delivery of signals will trigger a double free and hence a memory corruption which the attacker may use to override with own code.
- Valid code, impossible to avoid memory corruption with current state of the art tools.



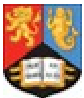
- Ask and rely on programmer's ability to write careful code and fully test each program before release.

```
sigtermhndlr(..);
sighuphndlr(..);

int main(void) {
   ...
   if((global_ptr1= calloc(..., sizeof(...)))==NULL)
      exit(1);

   if((global_ptr2= calloc(..., sizeof(...)))==NULL)
      exit(1);
   ...
   signal(SIGTERM, sigtermhndlr);
   signal(SIGHUP, sighuphndlr);
   ...
}
```

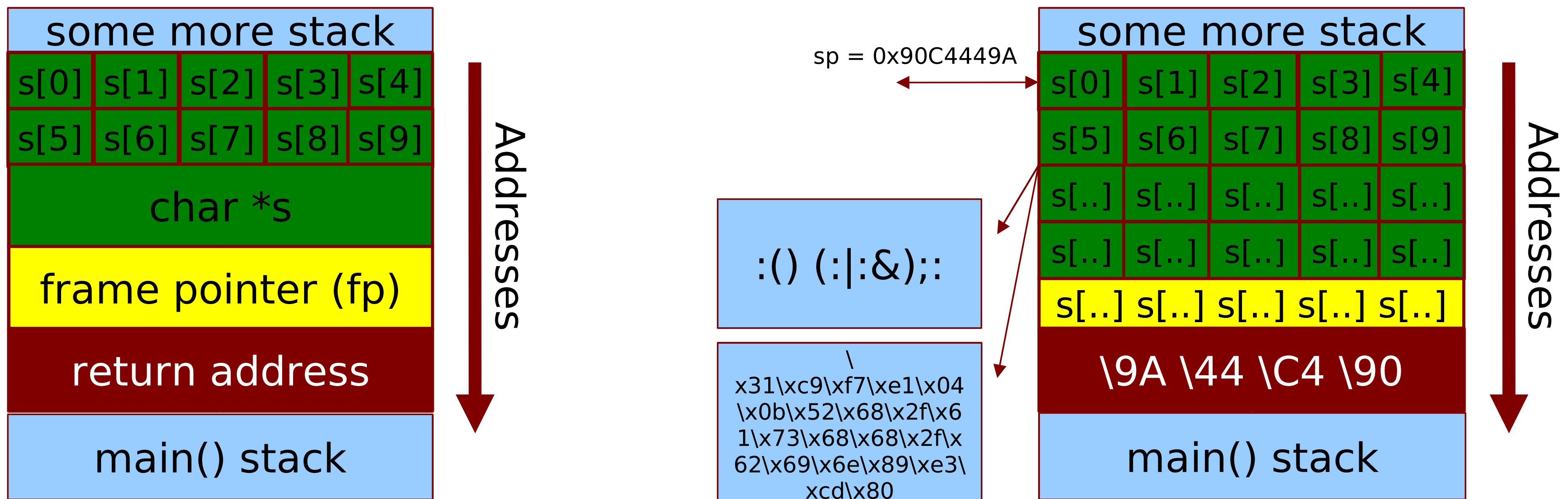
- Develop code-level dialect to solve problems.



```
void dodgy_boomer(char *data) {
   ...
   char *s;
   s = calloc(10, sizeof(char));

   /* bounds check? anybody? hello? */
   strcat(s, data)
   ...
}

void main(int argc, char *argv) { dodgy_boomer(argv[1]); }
```

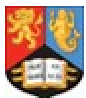


Execution before this line proceeded with super user privileges

```
...
dropped_permissions = true;
setuid(getuid());
...
```

Execution proceeds with normal user privileges.

- A frequent source of problems is the definition of “programmer-meaningful” variables (ie: flags).
- Code is valid in all cases and will be accepted by any compiler.
- The code before the breakpoint will be executed with super-user privileges and the code after the breakpoint will be processed with normal user privileges.
- Interrupting the program or diverting its execution exactly on the breakpoint will make the rest of the code believe that the permissions have already been dropped.



Cyclone

- Cyclone introduces regions, bounds checks, fat pointers, thin pointers, and generally solves most problems related to memory corruption.
- Cyclone can't solve the concurrency issue of the signal handler case.
- RIP Cyclone 2006!

Vault

- Vault introduces tracked variables and states.
- Vault can't solve the allocation problem directly but requires the programmer to modify a lot of code to catch the case in which memory could not be allocated.
- RIP Vault 2004!

- In order to concurrency errors which may lead to memory corruption we have to develop a “language feature” that implements pre- and postconditions.
- In order to avoid minor but frequent memory corruption problems which give a proud statistic of 46% of all Internet remote exploits we would need to implement a better memory handling model.
- To avoid atomic errors such as code interruptions we would need to implement states and tracked variables.
- We do NOT aim to force the implementation of this language on every piece of existent code but would consider porting parts (ie: libraries) which may affect a whole: