

# Learning Aims

- Determine which parts of a program can be tackled at the same time
- Determine the benefits and trade-offs of concurrent processing



- ‘Thinking concurrently’ is the need to work out which parts of a program can be developed to be processed **at the same time**, and which parts are dependent on other parts

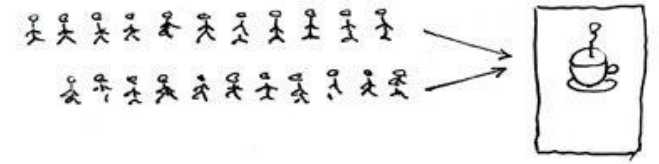
# Thinking Concurrently

- Any situation in the design or programming of a system when you would want more than one thing happening at the same time.
- Means programs need to be specifically designed to take advantage of this.
- Modules processed at the same time should be independent.
- Well-designed programs can save a lot of processing time.
- Programs have to be written specifically to take advantage of parallel processing which can make them longer and more complex.
- However, there is the issue of sequential tasks, **not all tasks can run concurrently**, which means sometimes there is no time saving as all the tasks still need to be run in order.

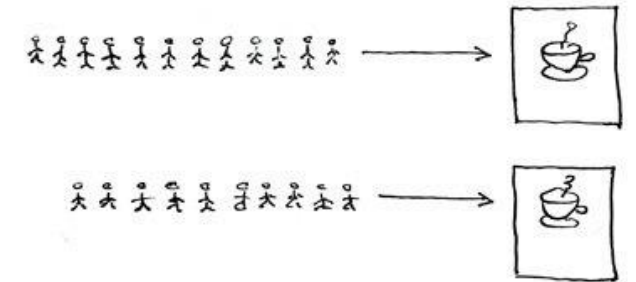
# Concurrent computing vs Parallel computing

- Generally, concurrent computing is defined as being related to but distinct from parallel computing.
- **Parallel computing** requires **multiple processors** each executing different instructions **simultaneously**, with the goal of speeding up computations.
- It is impossible on a single processor.
- **Concurrent processing**, on the other hand, takes place when several processes are running, with each in turn being given a **slice of processor time**. This gives the appearance that several tasks are being performed simultaneously, even though only **one processor** is being used. (Processor scheduling algorithms)

Concurrent = Two Queues One Coffee Machine



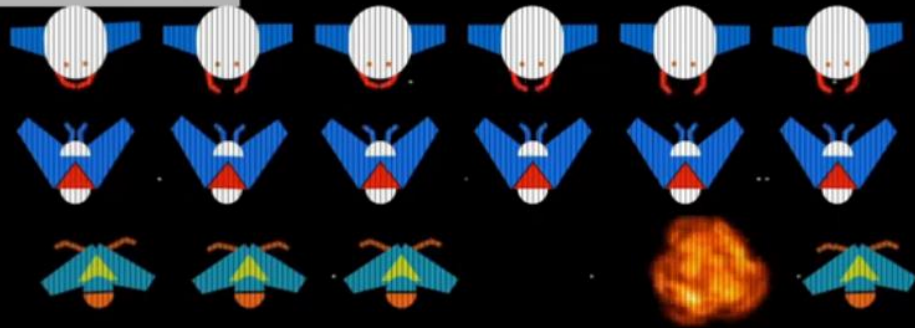
Parallel = Two Queues Two Coffee Machines



© Joe Armstrong 2013

## Being able to think concurrently

- Moving the player ship.
- Firing bullets.
- Bullet movement.
- Enemy ship movement.
- Updating players score.
- Collision detection.



- Consider this simple space shooter game.
- **Thinking concurrently**, what aspects of this problem could be developed to take place (be processed) at the same time?

# Concurrent computing and multithreading

- Parallel - Processes are happening at the same time only if multicore
- Concurrent - On a single core only 1 process can actually happen at a time, concurrent tries to **simulate multiple processes**. Processes can be in progress at the same time.
- Individual processes are **threads**, each thread has a life line.

# Multi-threading

## Being able to think concurrently

```
1295 this.formVisible.edit = true;
1296
1297 this.errors.clear('siteName');
1298 this.form.site = val;
1299
1300 },
1301 notesUpdating() {
1302   this.notesLoading = true;
1303 },
1304 notesUpdated() {
1305   getAppointment({
1306     id: this.form.id,
1307   }).then((res) => {
1308     this.$set(this.form, 'notes', res.data.notes);
1309     this.notesLoading = false;
1310     this.fetchAppointments().then(() => {
1311       this.prepareDataForBoard();
1312     });
1313   }).catch((error) => {
1314     this.notesLoading = false;
1315     if (!axios.isCancel(error)) {
1316       this.handleResponseError(error);
1317     }
1318   });
1319 },
1320 saveSubmit() {
1321   this.$refs['form'].validate((valid) => {
1322     if (valid) {
1323       this.formLoading = true;
1324
1325       let action = this.form.id ? editAppointment : addAppointment;
1326
1327       action(this.form).then((response) => {
1328         this.formLoading = false;
1329         this.$message({
1330           message: response.data.message,
1331           type: response.data.type
1332         });
1333         if (this.form.id) {
1334           if (response.data.type == 'success') {
1335             this.formVisible.edit = false;
1336           }
1337         }
1338         if (response.data.type == 'success') {
1339           this.formVisible.add = false;
1340         }
1341       });
1342     }
1343   });
1344 }
```

MORE VIDEOS



```
160 },
161 methods: {
162   ...methods,
163   async fetchConsultants() {
164     if (this.listSource) {
165       this.listSource.cancel('Fetch and cancel token');
166     }
167     this.listSource = CancelToken.source({
168       cancelToken: this.listSource.token
169     });
170     let params = {
171       search: this.filters.search,
172     };
173     this.loading = true;
174     return await getConsultants(params, {
175       cancelToken: this.listSource.token
176     }).then((res) => {
177       this.consultants = res.data;
178     }).finally(() => {
179       this.loading = false;
180       this.listSource = false;
181     });
182   },
183   applySearch: _.debounce(function (e) {
184     this.fetchConsultants();
185   }, 300),
186   clearSearch() {
187     this.filters.search = '';
188     this.fetchConsultants();
189   },
190   fetchSettings() {
191     getSettings({settings: Setting.COMPLIANCE}).then((res) => {
192       this.$set(this, 'settings', response.data);
193       if (Object.keys(this.settings).length) {
194         for (let timeName in this.timeSlotsData) {
195           let setting = _.find(this.settings, {
196             timeName: timeName,
197             (setting ? setting.value : null) : this.timeSlotsData[timeName]
198           });
199           this.$set(this, timeName, setting ? setting.value : null);
200         }
201       }
202     });
203   },
204   ability(row) {
205     return row.userId === this.userId ? 'canEdit' : 'canView';
206   }
207 }
```



- All the problems you have solved have probably been what we call single threaded programs.
- In other words you could actually put your finger on the code and trace it line by line, jumping off to **procedures** and **functions** and **branching** when required.
- It is however possible to write programs which have multiple threads.
- You can think of this as if you have multiple fingers tracing through your code at the same time.
- Each finger is following its own separate thread and is executing its code at the same time.

# Benefits and trade-offs of concurrent processing

- Concurrent processing has benefits in many situations.
- Increased program **throughput** – the number of **tasks** completed **in a given time** is increased
- Time that would be wasted by the processor **waiting** for the user to input data or look at output is used on another task
- The drawback is that If a large number of users are all trying to run programs, and some of these involve a lot of computation, these programs will take longer to complete



# Benefits and trade-offs of parallel processing

- Parallel processors enable several tasks to be performed simultaneously by different processors.
- It can speed up processing enormously when repetitive calculations need to be performed on large amounts of data
- Graphics processors can quickly render a 3-D object by working simultaneously on individual components of the graphic
- A browser can display several web pages in separate windows and one processor may be carrying out a lengthy search or query while processing continues in other windows

## **Parallel processing has limitations:**

- There is an overhead in coordinating the processors and some tasks may run faster with a single processor than with multiple processors.